Opiskelijoiden mentaaliset mallit ohjelmien suorituksesta ohjelmoinnin peruskurssilla

Vesa Vainio

Kognitiotieteen pro gradu -tutkielma
Psykologian laitos
Käyttäytymistieteellinen tiedekunta
Helsingin yliopisto
Elokuu 2006

HELSINGIN YLIOPISTO - HELSINGFORS UNIVERSITET - UNIVERSITY OF HELSINKI

Tiedekunta - Fakultet - Faculty	Laitos - Institution - Department
Käyttäytymistieteellinen tdk	Psykologian laitos

Tekijä - Författare – Author
Vesa Vainio

Työn nimi – Arbetets titel
Opiskelijoiden mentaaliset mallit ohjelmien suorituksesta ohjelmoinnin peruskurssilla

Title
Students's mental models about program execution on a fisrt programming course

Oppiaine - Läroämne - Subject
Kognitiotiede

Työn laji - Arbetets art - Level	Aika - Datum - Month and year	Sivumäärä - Sidoantal - Number of pages
Pro Gradu -tutkielma	Elokuu 2006	119 + 23 liitteitä

Tiivistelmä - Referat - Abstract

Mentaaliset mallit ovat mielensisäisiä tietoesityksiä, jotka mahdollistavat monimutkaista toimintaa koskevien päätelmien tekemisen. Tietokoneohjelmien suorituksen simulointi mielessä on opiskelijoille vaikeaa, mutta monessa tilanteessa myös edellytys onnistuneelle ohjelmien ymmärtämiselle ja suunnittelulle. Simulointikelpoisten mentaalisten mallien on todettu olevan mallin haltijalle itselleenkin epäselvästi määriteltyjä ja epästabiileja. Mentaaliset mallit voivat muodostaa hierarkkisia rakenteita, joissa alimmalla tasolla ovat konkreettiset käsiteltävät asiat, esimerkiksi ohjelmointikielen lauseet, ja korkeammilla tasoilla abstraktimmat kokonaisuudet, kuten algoritmit tai ohjelmiston osat. Mentaaliset mallit voivat sisältää kausaaliseen päättelyyn tarvittavaa toimintaan liittyvää tietoa sekä käyttötarkoitukseen liittyvää tietoa, kuten reunaehtoja sille, missä tilanteissa mallia voidaan käyttää. Luotettavasti onnistuvan päättelyn edellytykseksi on esitetty, että mallissa toimintaan ja käyttötarkoitukseen liittyvät tiedot eivät saa sekaantua keskenään. Tässä tutkimuksessa tarkasteltiin opiskelijoiden virhekäsityksiä, jotka liittyivät ohjelmien toimintaan, sekä opiskelijoiden perusteluja omille käsityksilleen. Mentaalisia malleja käsittelevä tutkimus antaa teoreettisen viitekehyksen tälle tarkastelulle.

Tutkimusta varten toteutettiin kuuden viikon haastattelusarja eksploratiivista puoli-strukturoitua haastattelumenetelmää käyttäen. Haastateltavat olivat ohjelmoinnin peruskurssin opiskelijoita ja haastatteluja tehtiin säännöllisesti koko peruskurssin keston ajan. Tutkimuksessa analysoitiin kolmen haastateltavan haastatteluja, kunkin haastatteluja 2–4 haastattelukerralta. Haastatteluista analysoitiin laadullista menetelmää käyttäen mentaalisiin malleihin liittyviä havaintokokonaisuuksia. Pääosin havaintokokonaisuudet olivat virheellisten mentaalisten mallien kuvauksia, joissa tuotiin esiin myös opiskelijoiden perustelut käsityksilleen. Saadut tulokset ovat yksityiskohtaisia kuvauksia yksittäisten opiskelijoiden ajattelusta. Raportoidut tulokset liittyvät olion, luokan ja tyypin käsitteiden ymmärtämiseen, olioiden viitesemantiikan ymmärtämiseen, opiskelijoiden mentaalisen simuloinnin abstraktiotasoon, päätelmien tekemiseen ohjelmakoodista, yhden esimerkin perusteella tehtyyn yliyleistämiseen ja muihin aiheisiin.

Tulosten merkitys on, että saaduista yksityiskohtaisista raporteista voidaan johtaa hypoteeseja tarkasteltavaksi laajempien otosten tutkimiseen soveltuvilla menetelmillä. Opetusta varten yksityiskohtaisia kuvauksia voidaan käyttää esimerkkeinä mahdollisista virhekäsityksistä, ja tutkimuskirjallisuudesta voidaan johtaa periaatteita vastaavien virhekäsitysten välttämiseksi.

Avainsanat – Nyckelord
mentaaliset mallit, mentaalinen simulointi, ohjelmoinnin psykologia, skeemat

Keywords
mental models, mental simulation, psychology of programming, schemas

Säilytyspaikka - Förvaringsställe - Where deposited
Käyttäytymistieteellisen tiedekunnan kirjasto, Helsingin yliopisto

Muita tietoja - Övriga uppgifter - Additional information

HELSINGIN YLIOPISTO - HELSINGFORS UNIVERSITET - UNIVERSITY OF HELSINKI

Tiedekunta - Fakultet - Faculty	Laitos - Institution – Department
Faculty of Behavioural Sciences	Department of Psychology

Tekijä - Författare – Author
Vesa Vainio

Työn nimi – Arbetets titel
Opiskelijoiden mentaaliset mallit ohjelmien suorituksesta ohjelmoinnin peruskurssilla

Title
Students's mental models about program execution on a first programming course

Oppiaine - Läroämne – Subject
Cognitive Science

Työn laji - Arbetets art - Level	Aika - Datum - Month and year	Sivumäärä – Sidoantal - Number of pages
Master's Thesis	August 2006	119 + 23 in appendice

Tiivistelmä - Referat – Abstract

Mental models are cognitive representations that enable inferences about complex activity. Mental simulation of execution of computer programs is difficult for students but in many cases it is also a requisite for successful comprehension and composition of programs. Runnable mental models are imprecisely specified and unstable even from the viewpoint of the person who holds the model. Mental models can form hierarchies where lowest levels are formed by concrete objects like statements of a programming language and higher levels are formed by more abstract entities such as algorithms or components of a software. Mental models can contain operational knowledge that enables causal reasoning or functional knowledge such as constraints for the application of the model. It has been proposed that a requisite for reliable reasoning is separation of structural and functional knowledge in the mental model. This thesis investigated students's thinking about program execution and those students's explanations for their own thinking. The study of mental models provides a theoretical framework for discussing observations about the students's thinking.

A series of interviews was conducted using an exploratory semi-structured interview format. Subjects were participants of a first programming course. Interviews were conducted repeatedly all through the duration of the course. Interviews from three subjects were analyzed with two to four interviews per subject. Interviews were qualitatively analyzed for coherent sets of observations about a particular theme. These themes were mostly about faulty mental models. Observations include the students's justifications for their thinking. The results are detailed descriptions of thinking processes of individual students. The reported results are related to students's understanding of the concepts of object, class and type, value and reference model of variables, level of abstraction of the mental simulation, making inferences about programs, overgeneralization based on one example and other themes.

The results can be applied for deriving hypotheses to be tested with methods more suitable for studying representative samples. The results can also be applied in programming pedagogics as examples of potential misunderstandings so that practices can be designed to avoid these misunderstandings.

Avainsanat – Nyckelord
mentaaliset mallit, mentaalinen simulointi, ohjelmoinnin psykologia, skeemat

Keywords
mental models, mental simulation, psychology of programming, schemas

Säilytyspaikka - Förvaringsställe - Where deposited
Library of Behavioural Sciences, University of Helsinki

Muita tietoja - Övriga uppgifter - Additional information

Sisältö

Kiitokset

Olen erittäin kiitollinen niille henkilöille, joilta olen saanut tukea tutkimusta tehdessäni. Kerätessäni aineistoa kesällä 2003 sain ystävällistä ja rohkaisevaa tukea lehtori Arto Wiklalta. Useaan otteeseen työni alusta loppuun asti professori Christina M. Krause on lukenut tekstiäni ja antanut siitä palautetta. Professori Jorma Sajaniemeltä olen saanut asiantuntevaa ohjausta erityisesti ohjelmoinnin psykologiaan liittyvien sisältökysymysten osalta keväästä 2004 lähtien työskentelyn loppuun asti. Kesällä 2005 tutkija Sami Surakka esitti luonnoksestani kommentteja. Keväällä 2006 tutkija Liisa Ilomäki antoi minulle arvokasta ohjausta erityisesti menetelmän ja analyysin raportointiin liittyen. Rakas avopuolisoni ja kognitiotieteen tutkija Nelli Salminen luki keväällä 2006 tekstiäni huolellisesti ja antoi paljon erityisesti kirjoitustekniikkaan liittyviä kommentteja ja oli muutenkin kiva ja söpö.

Helsingin yliopiston Tietojenkäsittelytieteen laitos tuki tutkielman tekoa stipendillä.

1. Johdanto

Jos henkilö osaa ohjelmoida tietokoneita jollakin ohjelmointikielellä, hänellä on silloin kyseiseen ohjelmointikieleen liittyvää tietoa. Tieto voi liittyä tietyn ohjelmointikielen rakenteisiin tai tietokoneohjelmien suoritukseen yleisesti. Lisäksi tällaisella henkilöllä on muistissaan tietoja aiemmin kirjoittamistaan tai lukemistaan ohjelmista. Hyvällä ja kokeneella ohjelmoijalla on käytössään laajemmat tiedot kuin vasta-alkajalla, mutta kokeneen ohjelmoijan tiedot ovat myös järjestyneet hänen mieleensä eri tavoin kuin vasta-alkajan tiedot (Adelson, 1981).

Kuten muitakaan ihmisen mielessä olevia tietoja, ohjelmoijien tietoja ei voida havainnoida suoraan ihmisen mielestä. Jos haluamme tutkia ohjelmoijan käyttämiä tietoja, on niitä koskevat havainnot tehtävä epäsuorasti havainnoimalla ohjelmoijan käyttäytymistä: vastauksia kysymyksiin, ääneenajattelua tai näppäilyä työskentelyn aikana, silmänliikkeitä ohjelman lukemisen aikana, kirjoitettuja ohjelmia tai muuta materiaalia. Nämä asiat voivat antaa tutkijalle tietoa siitä, millaista tietoa ohjelmoija käyttää ajattelussaan.

Monimutkaiseen asiaan, kuten ohjelmointiin, liittyvät tiedot osaajan mielessä eivät ole yksittäisiä ja irrallisia tiedonhippusia, joita voitaisiin tutkia yksitellen ja irrallaan, vaan tiedot ovat järjestyneet monimutkaisiksi kokonaisuuksiksi. Tällaisten monimutkaisten kokonaisuuksien *rakenteen* päätteleminen tutkijan käytössä olevin keinoin on erityisen haastavaa. Päättelyä voidaan tehdä esittämällä teorioita tai malleja tietojen rakenteesta ja suunnittelemalla koeasetelmia, joilla pyritään testaamaan, vastaako ohjelmoijien käyttäytyminen esitettyjä teorioita tai malleja. Näistä tutkimuksen kohteena olevista ihmisen mielessä olevista tietokokonaisuuksista käytetään muiden muassa nimiä *mentaalinen malli* tai *skeema*. Kumpikin termi on vakiintunut käyttöön, mutta samalla termit tulee ymmärtää jonkinlaisina yleisniminä mielensisäisille tietoesityksille, koska kummankaan rakenteesta tai käytöstä ei ole olemassa yksiselitteisiä tai yleisesti hyväksyttyjä määritelmiä.

Tässä tutkimuksessa tarkastellaan opiskelijoiden mentaalisia representaatioita, jotka liittyvät ohjelmien toimintaan. Mentaalinen tarkoittaa mielensisäistä ja representaatio tarkoittaa tietoesitystä. Tätä tutkimusta varten toteutettiin haastattelusarja, jossa haastateltavat olivat ohjelmoinnin peruskurssin opiskelijoita. Haastatteluissa käsiteltiin laajasti peruskurssin

käsitteistöön liittyviä asioita. Käytetty haastattelumenetelmä oli sellainen, että sen avulla saatiin syvälle luotaavaa tietoa haastateltujen tiedoista ja käsityksistä sekä *perusteluista*, joita opiskelijat esittivät käsityksilleen. Haastattelujen aihetta ei ollut ennalta rajattu johonkin tiettyyn teemaan tai käsitteeseen. Syvyyden vastapainona on, että tehtyjen havaintojen yleisyydestä ei nyt kerätyn ja analysoidun aineiston perusteella voida tehdä päätelmiä. Tämä tutkimus on siis eksploratiivinen tutkimus, joka pyrki dokumentoimaan ohjelmoinnin opiskelijoiden tietojen rakennetta ilman ennakko-oletuksia siitä, millaisia käsityksiä opiskelijoilta saatettaisiin havaita. Opiskelijoiden tietojen rakennetta selvitettiin teoreettisen psykologisen tiedon avulla sekä niiden perustelujen avulla, joita opiskelijat esittivät tiedoilleen. Tällaisen eksploratiivisen tutkimuksen tuottama tieto toimii pohjana muodostettaessa uusia teorioita tai malleja. Haastattelujen perusteella dokumentoitiin opiskelijoiden mentaalisia malleja. Lisäksi dokumentoitiin jonkin verran havaintoja siitä, millaisia strategioita opiskelijat käyttivät ohjelmia tarkastellessaan. Strategiat ovat mukana tarkastelussa, koska kuvaamalla strategiaa kuvataan samalla, miten mielessä olevaa tietoa käytetään tai muodostetaan. Tutkielman tulokset esitetään samaan aiheeseen liittyvinä havaintokokonaisuuksina. Tällaisia aiheita ovat esimerkiksi tyypin käsitteeseen liittyvät havainnot, olioiden viitesemantiikkaan liittyvät havainnot tai ohjelmien simuloinnin abstraktiotasoon liittyvät havainnot.

Tämä tutkielma kuuluu kognitiotieteen alaan eikä tietojenkäsittelytieteen alaan, joten tietokoneohjelmoinnin käsitteiden esittely ei ole osana tätä tutkielmaa. Lukijalta odotetaan ohjelmien ja ohjelmointikielten toiminnan tarkasteluun tarvittavien käsitteiden tuntemusta. Tämä on väistämätöntä, koska tutkielman aiheena on tarkastella näiden käsitteiden oppimista erittäin yksityiskohtaisella tasolla. Tarvittava käsitteistö sisältää olio-ohjelmointiin liittyvää käsitteistöä, kuten seuraavat käsitteet: muuttuja, tyyppi, olio, luokka, metodi, parametri, kapselointi, viitesemantiikka ja arvosemantiikka. Mainittujen käsitteiden tekemistä ymmärrettäväksi ohjelmointia tuntemattomalle lukijalle ei tässä edes yritetä, koska käsitteet ovat erittäin abstrakteja ja vaikeasti omaksuttavia. Tästä toimii todisteena jo se, että tätä tutkimusta varten haastatellut opiskelijat olivat kuunnelleet useita kokeneen opettajan pitämiä neljän tunnin luentoja aiheesta, eikä heidän ymmärryksensä käsitteistä silti läheskään aina ollut selkeää. Toisaalta käsitteiden omaksumisen vaikeus toimii hyvänä motivaationa niitä koskevien mielensisäisten tietoesitysten tutkimiselle.

Tämän tutkielman rakenne on sellainen, että tässä johdantoluvussa käsitellään laaja katsaus ohjelmoinnin aiempaan tutkimukseen, joka on käsitellyt mentaalisia malleja tai ohjelmoinnin psykologiaa, erityisesti aloittelevien ohjelmoijien kannalta. Johdatoluvun lopussa käsitellään tarkemmin tämän tutkimuksen tarkoitusta ja tavoitteita. Luvussa 2 käsitellään tässä tutkimuksessa käytetty haastattelumenetelmä ja luvussa 3 aineiston analysointi. Luvussa 4 raportoidaan tutkimukset tulokset havaintokokonaisuuksina, jotka liittyvät ohjelmointikielen käsitteiden ymmärtämiseen, kognitiivisiin prosesseihin sekä ulkoisten apuvälineiden käyttöön. Luvussa 5 on pohdinta, jossa käsitellään erikseen menetelmän pohdinta, tulosten pohdinta sekä ajatuksia tulevista tutkimussuunnista.

1.1. Mentaaliset mallit

1.1.1. Yleistä mentaalisista malleista

Tässä tutkimuksessa mentaalisella mallilla tarkoitetaan jotakin järjestelmää tai mekanismia koskevaa mentaalista representaatiota, joka mahdollistaa tarkasteltavan kohteen toiminnan ymmärtämisen, selittämisen ja ennustamisen. Mentaalinen malli saattaa sisältää mentaalisia kuvauksia esimerkiksi mekanismin tilojen tai toimintojen välisistä kausaalisista (syy- ja seuraussuhteisiin liittyvistä) suhteista tai muista dynaamisista piirteistä. Tietokoneohjelmien toiminta on juuri sellaista dynaamista toimintaa, jonka ymmärtäminen edellyttää tietoesityksiä, jotka mahdollistavat kausaalisten suhteiden esittämisen.

Norman esitti vuonna 1983 (Norman, 1983) tunnetun mentaalisten mallien luonnetta koskevan kuvauksen. Normanin havainnot perustuvat pitkäaikaiseen kokemukseen erilaisten järjestelmien ja laitteiden käytettävyyden tutkimuksesta. Normanin mukaan mentaalisilla malleilla on seuraavia piirteitä. Mentaaliset mallit ovat mallin haltijalle itselleenkin **epäselvästi määriteltyjä**. Henkilö saattaa itse olla epävarma siitä, mitä osia jonkin järjestelmän toiminnasta ymmärtää. Henkilö saattaa olla epävarma omasta ymmärryksestään myös silloin, kun ymmärrys itse asiassa on kattavaa ja oikeaa. Mentaaliset mallit ovat myös **epästabiileja**, ne voivat muuttua koska tahansa. Kysyttäessä henkilöltä hänen ymmärryksestään, henkilö saattaa kysymyksen seurauksena ruveta reflektoimaan omaa ymmärrystään

tavalla, jota ei ole aiemmin tehnyt, ja tämän seurauksena mentaalinen malli saattaa muuttua. Henkilön mentaaliset mallit saattavat olla myös **limittäisiä** siten, että esimerkiksi samankaltaisia järjestelmiä koskevat mentaaliset mallit voivat sekoittua keskenään.

Järjestelmä ja mekanismi voidaan ymmärtää laajassa ja abstraktissa merkityksessä. Järjestelmä voi tarkoittaa esimerkiksi fyysistä laitetta tai tietokoneohjelmistoa, tai näiden osaa. Mekanismi voi tarkoittaa mitä tahansa asiaa, jossa järjestelmällä on useita mahdollisia tiloja ja tilasta toiseen siirtyminen tapahtuu jonkin kausaalisen syyn johdosta. Fyysiset laitteet voivat toteuttaa mekanismeja. Klassinen esimerkki fyysisen laitteen toimintaan liittyviä mentaalisia malleja käsittelevästä tutkimuksesta on Kemptonin (1986) tutkimus, jossa tutkittiin lämmityslaitteen termostaatin toimintaan liittyviä mentaalisia malleja. Keskeinen järjestelmiä ja mekanismeja koskevia mentaalisia malleja käsittelevä teos on Gentnerin ja Stevensin toimittama kirja vuodelta 1983 (Gentner & Stevens, 1983).

Järjestelmien ja mekanismien toiminnan ymmärtämiseen liittyvä tutkimus on olennaista ohjelmoinnin kannalta, koska tietokoneohjelma on mekanismin kuvaus ja ohjelman kirjoittamisen tarkoitus on kuvata tietokoneelle ”ohjeet” siitä, miten tietoa pitäisi käsitellä. Tässä ohjelman käsittelemä tieto (eli *data*) muodostaa järjestelmän tilan ja tietokoneohjelman kuvaamat ”käsittelyohjeet” kausaalisen mekanismin, jonka mukaan tilasta toiseen siirtyminen tapahtuu.

Mentaaliset mallit mahdollistavat mentaalisen simuloinnin. Klein (1998, s. 52–53) on tutkinut mentaalisen simuloinnin merkitystä päätöksenteossa ja ongelmanratkaisussa. Kleinin mukaan mentaaliset simulaatiot ovat kohtalaisen yksinkertaisia, niissä on noin kolme tekijää (tai ”liikkuvaa osaa”) ja simulaatiossa käydään läpi enimmillään noin kuusi vaihetta tai tilaa. Syy näille rajoituksille on työmuistin rajallinen kapasiteetti. Kleinin mukaan hyödyllisten mentaalisten simulaatioiden muodostaminen ei ole helppoa, vaan siihen vaaditaan melko paljon kokemusta käsiteltävästä ongelma-alueesta. Kokemusta tarvitaan siihen, että osataan valita simulointiin mukaan juuri ongelman kannalta keskeiset tekijät ja valita simuloinnille oikea abstraktiotaso. Simuloinnin yrittäminen liian alhaisella abstraktiotasolla saa aikaan sen, että simulaatioon tulee mukaan liikaa tekijöitä tai vaiheita, eikä simulointi onnistu. Jos simulointia yritetään liian korkealla abstraktiotasolla, se ei tuota halutun ongelman ratkaisuksi sopivaa tietoa.

1.1.2. Mentaaliset mallit kirjallisuudessa

Tässä tutkimuksessa mentaalinen malli ymmärretään representaatioksi, joka mahdollistaa tarkasteltavan toiminnan ymmärtämisen, selittämisen ja ennustamisen. Termiä ”mentaalinen malli” käytetään tutkimuskirjallisuudessa myös muissa merkityksissä. Yksi selkeästi määritelty mentaalisten mallien tutkimuksen suuntaus tarkoittaa mentaalisilla malleilla päättelyn aikana työmuistiin muodostettavia lyhytaikaisia representaatioita. Johnson-Laird on esittänyt yksityiskohtaisen teorian (Johnson-Laird, 1983) näistä työmuistin mentaalisista malleista. Markman ja Gentner (2001) käyttävät Johnson-Lairdin mentaalisista malleista nimitystä loogiset mentaaliset mallit ja järjestelmiä ja mekanismeja koskevista mentaalisista malleista nimitystä kausaaliset mentaaliset mallit. Nimitykset kuvastavat hyvin sitä, että toisessa tapauksessa kyse on loogisesta päättelystä (joka on ajallisesti lyhytkestoinen prosessi, jota pidetään yllä työmuistissa) ja toisessa taas pitkän ajan kuluessa opitusta tiedosta, joka koskee jonkin mekanismin tai järjestelmän toimintaa määrääviä kausaalisia suhteita.

Termiä ”mentaalinen malli” on käytetty myös tekstinymmärrykseen liittyvissä tutkimuksissa. Tällöin mentaalista mallia käytetään synonyyminä tilannemallin (situational model) kanssa. Tilannemalli liittyy Van Dijk & Kintschin (1983; myös Kintsch, 1988) malliin tekstin ymmärtämisestä, jossa tekstistä muodostetaan kolmitasoinen representaatio. Alimmalla tasolla on sananmukainen, syntaktinen kuvaus tekstistä. Toisella tasolla on kuvaus tekstin sisältämistä merkityksellisistä propositioista. Nämä kaksi alinta tasoa vastaavat muodoltaan tekstin muotoa. Kolmas taso on tilannemallin taso, joka on malli tilanteesta, jota teksti kuvaa. Tilannemallin muodostaminen ei ole lukemisessa automaattinen toiminto, vaan se vaatii aikaa ja prosessointia.

Ohjelmoinnin psykologian yhteydessä mentaalisten mallien luonnetta on pohtinut Pennington (1987). Penningtonin tutkimuksessa ohjelmoijan tietoja tietystä ohjelmasta kuvattiin eräänlaisella jakaumalla. Jakauma muodostui viidestä mahdollisesta kategoriasta, joista kukin kuvasi yhtä tiedon lajia, joka ohjelmoijalla voi ohjelmasta olla tutustuttuaan ohjelmaan. Kategoriat olivat operaatiot (operations), ohjausvuo (control flow), tietovuo (data flow), tila (state) ja funktio (function). (Kumpikin kategorioista ”operations” ja ”function” olisi mahdollista suomentaa toiminnoiksi, mutta sekaannusten välttämiseksi

näin ei tehdä. Operaatiot tarkoittavat matalan tason toimintoja, kuten yksittäisten arvojen sijoituksia muuttujan arvoksi. Funktiota koskeva tieto käsittelee tarkoitusta eli tietoa, miksi jokin asia tehdään.) Penningtonin tutkimuksessa ohjelmaa koskevaa ymmärrystä kuvattiin jakaumalla, jossa jokaisen kategorian tietoja kuvattiin suhdeluvulla, joka ilmaisi, miten suureen osuuteen kyseisen kategorian tietoja testaavista kyllä/ei-kysymyksistä henkilö oli vastannut oikein.

Tämän lisäksi Pennington tutki ohjelmoijien tietoja ohjelmista kirjoitettujen yhteenvetojen ja ääneenajatteluprotokollien perusteella. Näistä tarkasteltiin sitä, esittivätkö ohjelmoijat toteamuksia, jotka liittyivät ohjelmaan itseensä, vai toteamuksia, jotka liittyivät ohjelman sovellusalueeseen eli niihin todellisen maailman asioihin, joiden käsittelyyn ohjelma oli tarkoitettu. Penningtonin tutkimuksessa ohjelmaa koskevalla tiedolla oli siis kaksi ulottuvuutta, joista toinen oli tiedon laji ja toinen tiedon ”yksityiskohtaisuuden taso”, joka vaihteli ohjelmaa koskevan ja sovellusaluetta koskevan tiedon välillä. Pennington piti malliaan analogisena mainitulle Van Dijkin ja Kintschin mallille siten, että ohjelmaa koskeva tieto on ”tekstipohjainen” malli ja sovellusaluetta koskeva tieto on Van Dijkin ja Kintschin termein tilannemalli. Penningtonin tulosten mukaan erityisen tehokas ymmärtämisstrategia on sellainen, jossa ymmärtäjä muodostaa yhteyksiä ohjelmaa koskevien ja sovellusaluetta koskevien tietojen välille.

Penningtonin tutkimuksessa (ja useissa muissa ohjelmoinnin psykologian tutkimuksissa) käsitellyt sovellusaluetta koskevaan tietoon liittyvät kysymykset ovat olennaisia tutkittaessa ohjelmoinnin ammattilaisia. Ohjelmoinnin aloittelijoita tutkittaessa sovellusaluetta koskevan tiedon merkitys on pienempi. Ensinnäkin ohjelmoinnin aloittelijoille jo pelkästään ohjelmaa koskevan tiedon muodostaminen ja käsittely on niin haastavaa, että se vie kaiken huomion. Toinen syy on, että peruskursseilla käsiteltävät ohjelmaesimerkit ovat niin yksinkertaisia, ettei niistä yleensä voida johtaa mitään merkityksellisiä yhteyksiä mihinkään todellisen maailman ilmiöihin.

Pennington pohti tutkimuksessaan, tarkoittaako ohjelmasta muodostettu mentaalinen malli ohjelmaa koskevaa tietoa vai sovellusaluetta koskevaa tietoa. Penningtonin kanta oli, että molempia voidaan kutsua mentaaliseksi malliksi. Lisäksi Pennington pohti, mikä piirre yleensäkään erottaa mentaalisen mallin muista mielensisäisistä tietoesityksistä, ja esitti yhdeksi erottavaksi tekijäksi sen, että mentaaliset mallit olisivat ”suoritettavia” (runnable).

Suoritettavuus edellyttäisi mielensisäistä tietoesitystä, joka pystyy esittämään kausaalisia suhteita. Tapa, jolla Penningtonin tutkimuksessa kuvataan ohjelmoijien tietoja eräänlaisina jakaumina, ei ota kantaa siihen, miten mentaalisessa tietoesityksessä voitaisiin esittää kausaalisia suhteita. Seuraavassa luvussa esiteltävä de Kleerin ja Brownin teoria sen sijaan on yksityiskohtainen teoria siitä, millaisista tietoesityksistä ”suoritettavat” mentaaliset mallit voisivat muodostua.

1.1.3. Suoritettavat mentaaliset mallit

De Kleer ja Brown (1981, 1983) esittivät teorian mentaalisista malleista, jotka mahdollistavat mekanismien mentaalisen simuloinnin. He kuvasivat yksityiskohtaisesti prosessia, jonka kautta mekanistisen laitteen rakenteen kuvauksen perusteella voidaan päätellä laitteen kausaalinen toiminta eri tilanteissa. De Kleer ja Brown kehittivät teoriansa käyttäen esimerkkinä sähkömekaanisia laitteita koskevaa ymmärrystä. Heillä ”laitteen rakenteen kuvaus” tarkoittaa esimerkiksi kytkentäkaaviota tai vastaavaa teknistä esitystä jonkin laitteen rakenteesta. Kausaalisen toiminnan päätteleminen eri tilanteissa tarkoittaa ensinnä sitä, että rakenteen kuvauksen perusteella pystytään päättelemään laitteen toiminta normaalissa tilanteessa. Lisäksi tämä tarkoittaa sitä, että jos laitteesta muodostettu ymmärrys on *robustia* eli tukevaa ja luotettavaa, voidaan ymmärryksen perusteella päätellä laitteen toiminta myös poikkeuksellisessa tilanteessa eli esimerkiksi laitteen jonkin osan rikkoutuessa. Teoriaa voidaan soveltaa myös ohjelmoinnin tarkasteluun. Tällöin rakenteen kuvaukseksi ymmärretään tietokoneohjelman lähdekoodi. Toiminnan päättelemiseksi ymmärretään sen päätteleminen, mitä ohjelma tekee, kun se annetaan tietokoneen suoritettavaksi. Tällaisen päättelyprosessin kuvaamisen edellytys on kuvata, millaista prosessin käyttämä tieto on rakenteeltaan ja ominaisuuksiltaan, sekä miten tätä tietoa käytetään.

De Kleerin ja Brownin teoriaa ei ole juurikaan mainittu ohjelmoinnin tutkimuksen yhteydessä ja muutoinkin teoria on jäänyt alansa kirjallisuudessa vähälle huomiolle. Ainoa maininta teoriasta on Adelsonin ja Solowayn (1985) tutkimuksessa, jossa sitä käytetään perustelemaan väitteitä, että simulointikelpoisen mentaalisen mallin muodostaminen ei onnistu, ennen kuin simuloitava asia on kuvattu riittävällä tarkkuudella ja että globaalin mallin muodostaminen riippuu sovellusaluetta koskevasta kokemuksesta. De Kleerin ja Brownin

teoria tarjoaa uskottavaa tukea myös näillä väitteille, vaikka teorian näitä osia ei tässä tutkimuksessa käsitelläkään.

De Kleerin ja Brownin käyttämä menetelmä on mentaalisten mallien simulaatioiden toteuttaminen. Tämä tarkoittaa kysymyksenasettelua ”Jos laitteen toiminnasta on käytettävissä tietyt tiedot, niin millaisia päätelmiä laitteen toiminnasta on mahdollista tehdä?” Jos simulaation tuottamat päätelmät eivät vastaa päätelmiä, joita ihmiset ilmiselvästi pystyvät tekemään, käytössä olleet tiedotkaan luultavasti eivät vastaa ihmisten käytettävissä olevia tietoja. Jos taas päätelmät vastaavat ihmisten tekemiä päätelmiä, voidaan olettaa, että kyseiset tiedot ovat realistinen kuvaus siitä, mitä tietoja ihmisillä on käytettävissään. Soveltamalla tällaista kysymyksenasettelua monipuolisesti erilaisiin päätelmiin, voidaan saada käsitys käytettävissä olevien tietojen rakenteen vaikutuksesta siihen, millaisia päätelmiä tietojen perusteella on mahdollista tehdä. Seuraavassa kuvataan, millaisten tietojen perusteella de Kleerin ja Brownin mukaan ihmiset tekevät päätelmiä laitteiden toiminnasta.

De Kleerin ja Brownin teoriassa laitteen mentaalinen malli rakentuu niin sanotuista **komponenttimalleista**. Komponentit tarkoittavat laitteen osia, ja komponenttimalli rakentuu säännöistä, jotka kuvaavat komponentin käyttäytymisen. Säännöt kuvaavat sen, millaisia tiloja komponentilla voi olla, millaiset tilasiirtymät ovat mahdollisia sekä miten komponentti vaihtaa informaatiota toisten komponenttien kanssa. Komponenttien väliset **yhteydet** välittävät informaatiota komponentilta toiselle. Kaikki komponenttien väliset yhteydet kuvataan samalla tavoin riippumatta niiden fyysisestä toteutuksesta. Fyysisten laitteiden tapauksessa yhteyksiä ovat esimerkiksi johtimet tai magneettikentät, tietokoneohjelmien tapauksessa esimerkiksi tietovuo ja ohjausvuo ohjelman osien välillä.

Mentaalisia malleja muodostetaan **kvalitatiivisen simulointiprosessin** avulla, jossa käytetään hyväksi tietoa laitteen rakenteesta sekä laitteen komponenttien toiminnasta (eli komponenttimallien luokkakohtaisia ominaisuuksia tai prototyyppejä). Mentaalinen malli kuvaa laitteen kausaalisuuden, eli sen perusteella pystytään päättelemään, mikä tila seuraa jostakin toisesta tilasta. Päättelyprosessi, jonka avulla mentaalista mallia käytetään, on myös simulointiprosessi. Mekanistiset mentaaliset mallit ovat siis **simulointikelpoisia mentaalisia malleja**. Mentaalisen mallin perusteella pystytään päättelemään, **miten** laite toimii ja **miksi** se toimii niin kuin toimii.

Seuraavassa esitellään periaatteita, joita noudattava mentaalinen malli on myös **robusti** (tukeva, luotettava), se pystyy ennustamaan laitteen toiminnan oikein myös poikkeuksellisissa tilanteissa. Poikkeuksellisia tilanteita ovat esimerkiksi tilanteet, joissa jokin komponentti rikkoutuu ja alkaa toimia poikkeavasti, tai joissa laitteen rakenteeseen tehdään muutos. Alkuperäisen teorian robustiusperiaatteista esitellään tässä vain kaksi.

Mentaalisen mallin robustin toiminnan kannalta tärkein periaate on **no-function-in-structure-periaate** eli ”pidä toiminta ja rakenne erillään” -periaate. Sen mukaan komponentin toimintaa kuvaavat säännöt eivät saa millään tavalla (edes implisiittisesti) viitata siihen, miten koko järjestelmän oletetaan toimivan. Jos periaatetta rikotaan, eli komponenttimallien kuvaamiseen käytetään sääntöjä, jotka viittaavat koko laitteen toimintaan, mentaalinen malli ei ole robusti, eikä pysty ennustamaan oikein laitteen toimintaa poikkeavissa tilanteissa.

Toinen robustiusperiaate on **paikallisuusperiaate**. Se on syntaktinen rajoite siitä, että komponenttimallin säännöt eivät saa viitata muihin kuin komponenttimallin paikallisiin arvoihin, joita ovat komponentin sisäinen tila sekä sen omat yhteydet muihin komponentteihin. Paikallisuusperiaate on tärkeä komponenttimallien yleisyyden kannalta. Jos sitä ei toteuteta, komponenttimallit ovat päteviä vain sen laitteen kokoonpanossa, jossa ne on määritelty, mutta niitä ei voida käyttää hyväksi toisissa kokoonpanoissa.

Seuraavassa on esimerkki robustiusperiaatteiden soveltamisesta ohjelmoinnin yhteyteen. De Kleer ja Brown käyttivät teoriansa yhteydessä esimerkkejä, jotka liittyivät sähkömekaanisten laitteiden toimintaan. Esiteltävä esimerkki on kirjoittajan suunnittelema. Käsiteltäessä for-lauseen toimintaa, voitaisiin käsitellä esimerkkilausetta:

```
for (i=0; i<10; i++) {
      System.out.println(i);
}
```

Lausetta voidaan kuvata seuraavasti: ”Lauseen suorituksen aluksi lauseen sisällä käytetty muuttuja i nollataan. Tämän jälkeen lause iteroi läpi i:n arvot nollasta yhdeksään ja tulostaa jokaisen arvon.” Kuvaus on intuitiivisesti täysin mahdollinen ja opetustilanteeseen sopiva. Kuvauksen perusteella oppija saattaa muodostaa yleisen for-lauseita koskevan käsityksen, että ”Aina for-lauseen suorituksen aluksi lauseen sisällä käsiteltävä muuttuja nollataan.”

Käsitys kuvaa esimerkin for-lauseen toiminnan oikein, samaten useiden muiden tyypilliseen tapaan kirjoitettujen for-lauseiden. Jos käsitystä tarkastellaan osana yleisesti for-lausetta kuvaavaa mentaalista mallia, käsitys ei ole robusti, sillä se rikkoo edellä mainittuja periaatteita. No-function-in-structure-periaatetta kuvaus rikkoo viittaamalla lauseen toimintaan (vaikka *usein* for-lauseen alustuksessa nollataan jokin muuttuja, nollaaminen ei ole osa for-lauseen *rakennetta*). Paikallisuusperiaatetta käsitys rikkoo viittaamalla samalla kertaa sekä for-lauseeseen että sen sisällä olevaan lauseeseen. Tarkastellaan, mitä tapahtuu, jos käsitystä sovelletaan seuraavaan esimerkkiin:

```
int luku = 64;
int laskuri = 0;

for (int l = 0; luku > 2; luku /= 2) {
      for (int k = 0; k < luku; k++) {
            laskuri++;
            System.out.println(laskuri);
      }
}
```

Tässä esimerkissä for-lauseen sisällä käsitellään *eri* muuttujaa kuin for-lauseen alustusosassa. Käsityksen ”Aina for-lauseen suorituksen aluksi lauseen sisällä käsiteltävä muuttuja nollataan.” perusteella pääteltäisiin, että aina sisemmän for-lauseen suorituksen aluksi muuttuja `laskuri` nollattaisiin. Tämä virhepäätelmä syntyy, koska osaksi for-lauseen *rakennetta* koskevaa ymmärrystä on omaksuttu käsitys, että for-lauseen sisällä käytetty muuttuja nollataan for-lauseen aluksi. Esimerkissä käytetty jälkimmäinen koodinäyte oli käytössä tutkielmaa varten tehdyssä haastattelussa, ja tutkielman tuloksissa (luvussa 4.3.5) nähdään esimerkki juuri tällaisesta virhekäsityksestä.

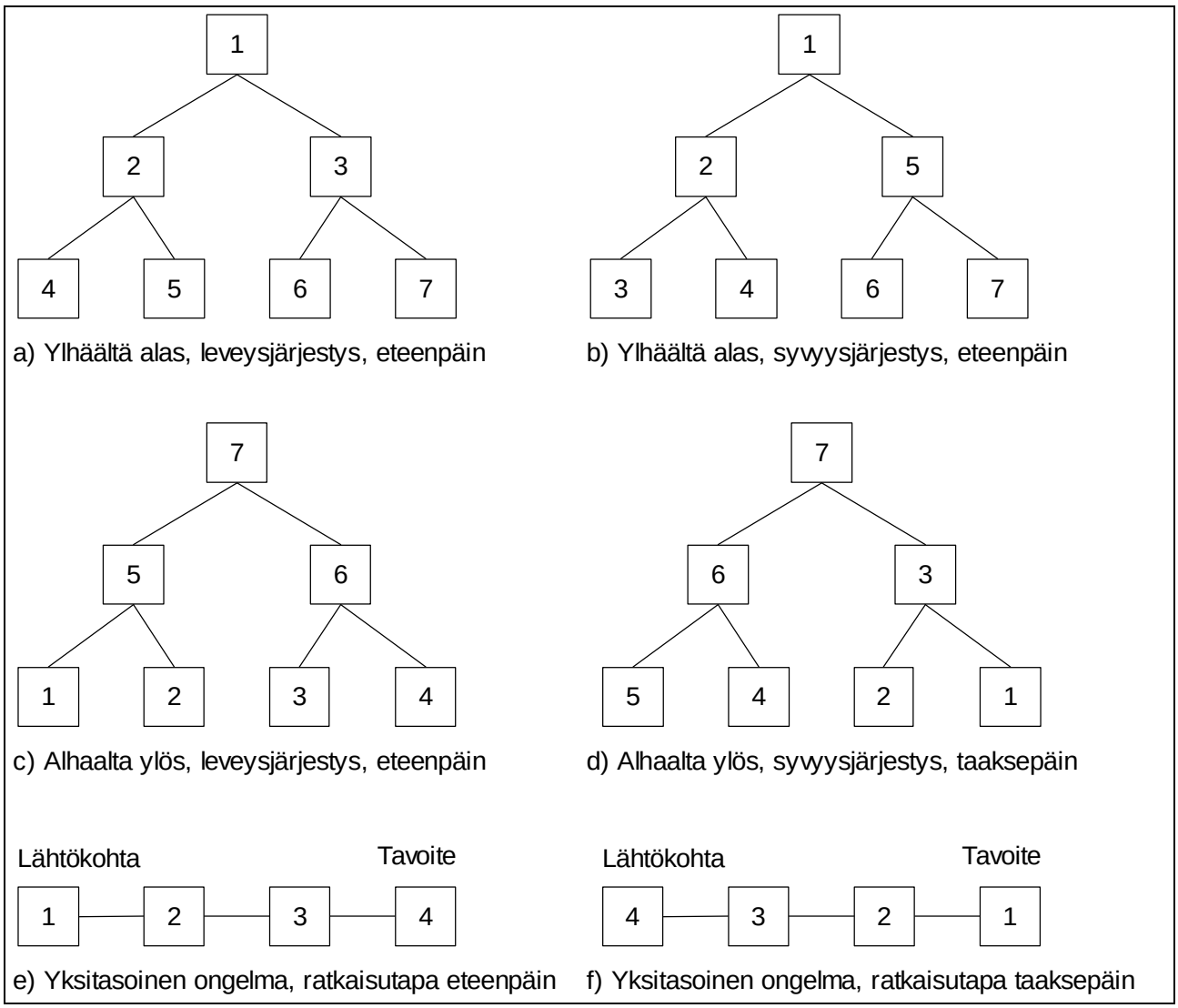
1.2. Ongelmanratkaisun tarkasteleminen

Ohjelmointi on ongelmanratkaisua: ohjelmia kirjoitetaan ratkaisuiksi ongelmiin. Ratkaisua muodostaessaan ohjelmoija joutuu yleensä koostamaan ratkaisunsa ohjelmointikielen ilmauksista tavalla, joka on entuudestaan tuntematon. Ongelmanratkaisua voidaan kuvata useilla erilaisilla käsitteillä. Järjestystä, jossa ongelman osat käsitellään, voidaan kuvata seuraavilla kolmella käsiteparilla (Détienne, 2002, s. 27):

- Ylhäältä alas / alhaalta ylös *(top-down / bottom-up)*
- Leveysjärjestyksessä / syvyysjärjestyksessä *(breadth-first / depth-first)*

- Etuperin / takaperin *(forward / backward)*

Ylhäältä alas ja alhaalta ylös viittaavat ongelman eri abstraktiotasoihin siten, että abstraktin tason ajatellaan olevan ylhäällä ja yksityiskohtaisen, konkreettisen tason ajatellaan olevan alhaalla. Lähes kaikkien ongelmien ratkaisu voidaan ajatella joko abstraktilla tasolla, jolla esitetään ratkaisun tarkoitus tai päämäärä, sekä konkreettisella tasolla, jolla esitetään ne konkreettiset toimet tai vaiheet, joiden avulla päämäärään päästään. Monimutkaisten ongelmien tapauksessa saattaa olla mahdollista erottaa ratkaisusta useita abstraktiotasoja. Kuvan 1 tilanteet a ja c havainnollistavat ylhäältä alas- ja alhaalta ylös -tapojen eroa.



Kuva 1. Ongelmanratkaisun mahdollisia etenemistapoja. Kussakin kohdassa ratkaisun ongelmaan ajatellaan koostuvan kaikista numeroiduista laatikoista yhteensä. Ratkaisua **toteutettaessa** askelten järjestys on kaikissa tilanteissa vasemmalta oikealle, mutta ratkaisua **suunniteltaessa** järjestys on kuvan numeroiden mukainen.

Jos ongelmanratkaisulla on useita abstraktiotasoja ja ratkaisu etenee ylhäältä alas, voidaan kysyä, eteneekö ratkaisu leveysjärjestyksessä vai syvyysjärjestyksessä. Tässä yhteydessä ratkaisua ajatellaan puumaisena rakenteena, jossa kullakin ratkaisun osalla voi olla yksi tai useita alakohtia seuraavaksi alemmalla abstraktiotasolla. Jos ratkaisu etenee leveysjärjestyksessä, kaikki samalla abstraktiotasolla olevat ratkaisun osat käsitellään ennen seuraavaksi alemmalle abstraktiotasolle siirtymistä. Jos taas ratkaisu etenee syvyysjärjestyksessä, kukin käsiteltäväksi otettu ratkaisun osa käsitellään alimmalle tasolle saakka ennen seuraavaan ratkaisun osaan siirtymistä. Kuvan 1 tilanteet a ja b havainnollistavat leveysjärjestyksen ja syvyysjärjestyksen eroa.

Ongelman ratkaiseminen etuperin tai takaperin liittyy siihen, missä järjestyksessä ratkaisuun tarvittavia toimia tai työvaiheita tuotetaan. Tuotettaessa ratkaisua etuperin lähdetään liikkeelle alkutilasta ja tuotetaan järjestyksessä vaiheita, joiden kautta päästään lopputilaan. Takaperin ratkaisua tuotettaessa lähdetään liikkeelle halutusta lopputilasta ja tuotetaan takaperin askeleita, joiden kautta lopulta alkutilasta voidaan päästä lopputilaan.

Useiden tässä tutkielmassa esitettyjen teorioiden ja tarkastelujen kannalta erityisen olennaista on ajatus siitä, että ongelmasta voidaan tehdä mielensisäinen esitys, jonka rakenne on hierarkkinen. Ongelma voidaan siis jakaa hierarkkisiksi osaongelmiksi, tai eritellä tavoitteiksi ja keinoiksi päästä tavoitteisiin. Puhuttiinpa kulloinkin skeemoista taikka mentaalisista malleista ongelmanratkaisun välineinä, olennaista tarkasteltaville asioille on, että niiden pitää pystyä liittymään toisiinsa siten, että ne muodostavat hierarkian.

1.3. Ohjelmoijien tieto

1.3.1. Ohjelmointiskeemat

Ongelmanratkaisun tutkimus on osoittanut, että keskeinen osa ongelmanratkaisua on ongelman hahmottaminen niin sanottujen skeemojen avulla, jotka ovat mentaalisia representaatioita ongelmiin opituista stereotyyppisistä ratkaisumalleista tai ratkaisujen osista (Saariluoma, 1988). Tällaisten ratkaisuskeemojen keskeinen rooli myös ohjelmoijien

ongelmanratkaisussa on osoitettu moneen kertaan ja eri tavoin (Brooks, 1977; Soloway & Ehrlich, 1984; Rist, 1986, 1989). Stereotyyppisten ratkaisuskeemojen hyödyllisyys aloittelijoiden tekemien virheiden analyysissä on myös osoitettu (Soloway, Bonar & Ehrlich, 1983; Spohrer, Soloway & Pope, 1985).

Eri tutkimuksissa ohjelmointiskeeman käsite määritellään eri tavoin, ja käsitteestä myös käytetään eri nimiä. International Journal of Man-Machine Studies -lehden ohjelmoinnin psykologiaa käsitelleen erikoisnumeron johdannoksi kirjoittamassaan artikkelissa Brooks (1990) viittaa neljään samassa numerossa ilmestyneeseen tutkimukseen ja toteaa, että yhdessä puhutaan suunnitteluskeemoista (design schemas), jotka määrittävät järjestelmän funktionaalisen dekomposition osajärjestelmiksi. Kolme muuta tutkimusta käyttävät termiä suunnitelma (plan), mutta eroavat toisistaan esimerkiksi sen suhteen, millaisen koostamismekanismin avulla skeemojen oletetaan liittyvän toisiinsa.

Eräs ohjelmoinnin psykologian tutkimussuuntaus, jossa keskeisenä hahmona on ollut Elliot Soloway, käsittelee ratkaisuskeemoja ohjelmiin liittyvinä **tavoitteina** (goals) ja **suunnitelmina** (plans). Tavoitteella tarkoitetaan ohjelmoijan ajattelussa esiintyvää tapaa ilmaista, mikä on ohjelman kirjoittamisen tavoite. Tavoitteet jakautuvat osatavoitteisiin, joista kukin ilmaisee päätavoitteen jonkin osan. Tavoitteet konkretisoituvat suunnitelmina. Suunnitelma ilmaisee, millä konkreettisilla toimilla tavoitteeseen päästään ohjelmassa. Korkean tason suunnitelmat voivat aiheuttaa uusia osatavoitteita, joiden toteuttamiseen tarvitaan matalamman tason suunnitelmia. Tässä yhteydessä suunnitelmilla tarkoitetaan kognitiivisia mieltämisyksiköitä, joilla ei ole mitään yhteyttä ohjelmistotuotannossa käytettyyn suunnitelman käsitteeseen, joka liittyy ennen ohjelmiston toteuttamista tehtävään suunnitelmaan ohjelmistosta (esimerkiksi Sommerville, 2001). Tavoitteet ja suunnitelmat voidaan nähdä niin, että ne muodostavat **hierarkkisen rakenteen**, jossa jokaista tavoitteita tarkennetaan joko jakamalla se osatavoitteisiin tai esittämällä suunnitelma, jolla tavoitteeseen päästään (Spohrer, Soloway & Pope, 1985; Soloway, 1986). Alimman tason suunnitelmat voidaan konkretisoida ohjelmakoodina, jolloin ne vastaavat yhtä tai muutamaa ohjelmointikielen lausetta. Tällaisen tavan käytöstä ohjelmien analysoinnissa on julkaistu useita elegantteja esimerkkejä (Soloway, 1986; Soloway & Ehrlich, 1984; Soloway, Ehrlich & Black, 1983; Soloway, Ehrlich & Bonar, 1982; Spohrer, Soloway & Pope, 1985; Soloway, Bonar & Ehrlich, 1983; Spohrer & Soloway, 1986; Détienne & Soloway, 1990).

Rist (1986, 1989) käsittelee suunnitelmien ja tavoitteiden suhdetta hieman eri tavalla. Suunnitelmia hän käsittelee stereotyyppisinä ohjelman osina, joita kertyy muistiin oppimisen ja kokemuksen tuloksena. Tavoitteet taas ovat tapa ketjuttaa *yhdessä ohjelmassa* esiintyvät suunnitelmat toisiinsa. Suunnitelmat siis ovat osa ohjelmoijan yleistä taituruutta, mutta tavoitteet ovat yhdestä tietystä ohjelmasta (joko suunniteltavasta tai luettavasta ohjelmasta) muodostettavan tietoesityksen ominaisuus. Mayer (1979) esittää ohjelmointia koskevan tiedon toistensa päälle rakentuvina tasoina. Hänen esityksessään mainitaan peräti kahdeksan eri tasoa. Brooks (1983) esittää, että ohjelman ymmärtämisen prosessi perustuu monitasoisen hierarkkisen representaation muodostamiseen, jonka avulla sovellusalueen käsitteet yhdistetään välitasojen kautta ohjelman toteutuksen osiin. Yhteistä kaikille tässä luvussa mainituille kuvauksille on, että ohjelmointitieto kuvataan monitasoisena hierarkiana, jossa ylemmät tasot esitetään alemmista tasoista koostuvien yksiköiden avulla. Voidaan siis yleisesti todeta, että **ohjelmointiskeemat kuvaavat ohjelmia usealla eri abstraktiotasolla ja muodostavat keskenään hierarkkisen rakenteen**.

Adelson ja Soloway (1985) totesivat eksperttien suunnittelukäyttäytymistä tutkiessaan, että jos ongelman kuvaus ei ole riittävän selvästi rajattu, ekspertit keskittyvät ongelman kuvauksen rajaamiseen. Tämä selittyy sillä, että ilman soveltuvia reunaehtoja ekspertit eivät voi valita, mitä skeemoja he suunnitelmaansa soveltavat. Détienne ja Soloway (1990) tunnistivat eräänä ohjelmoijien käyttämänä strategiana strategian ”Skeemojen reunaehtojen käyttäminen päättelyssä”, jota käytetään juuri pyrittäessä tunnistamaan, mitä skeemaa parhaillaan luettava koodi vastaa. Voidaan siis todeta, että **ohjelmointiskeemat sisältävät reunaehtoja skeemojen itsensä käytölle**.

Adelson ja Soloway (1985) totesivat, että ohjelmien simulointia tehdään vain silloin, kun ennestään tuttuja skeemoja liitetään toisiinsa tavalla, joka ei ole ennestään tuttu. Simulointia ei siis tehdä, jos käytössä ei ole ennestään tuttuja skeemoja lainkaan, tai jos tapa, jolla skeemat liitetään toisiinsa, on tuttu. Ohjelmien lukemisen osalta Détienne ja Soloway (1990) tunnistivat konkreettisen simuloinnin strategian, jota käytetään juuri silloin, kun tuttuja skeemoja on liitetty toisiinsa ennestään tuntemattomalla tavalla. Näistä havainnoista syntyy vaikutelma, että **skeemoihin voi sisältyä ohjelman toimintaan liittyvää kausaalista tietoa, joka liittää skeeman siihen kontekstiin, jossa sitä käytetään ohjelmassa**. Jos skeema on tutussa kontekstissa, simulointia ei tarvita, koska tieto kausaalisista vuorovaikutuksista on koodattuna skeemaan. Sen sijaan jos skeema esiintyy vieraassa konteks-

tissa, ymmärrys kausaalisista suhteista on muodostettava simuloimalla, jotta mahdolliset ennestään tuntemattomat vuorovaikutukset tulevat selvitetyiksi.

Nuorilla vasta-alkajilla tehdyt tutkimukset (Perkins & Martin, 1986; Perkins *et al.*, 1986) viittaavat siihen, että aloittelijoilla yksittäisten lauseiden välisten kausaalisten suhteiden ymmärtäminen ei onnistu ilman huolellista simulointia. Vasta-alkajilla yksittäisiä ohjelmointikielen lauseita koskevat skeemat eivät vielä ole täysin kehittyneitä eli eivät sisällä kattavia tietoja lauseiden kausaalisista yhteyksistä. Tämä vahvistaa käsitystä siitä, että ohjelmointiskeemat sisältävät tietoa kausaalisista yhteyksistä.

Useimmat skeemoja käsitelleet tutkijat eivät juurikaan ota kantaa siihen, millaisen prosessin kautta skeemoja opitaan. Ohjelmoinnin psykologiassa poikkeus tähän on Ristin (1989) tutkimus, jota käsitellään myöhemmin. Muutoin kuin ohjelmoinnin tutkimuksen yhteydessä Anderson on esittänyt laskennallisia malleja skeemojen käytöstä ja oppimisesta. Neves ja Anderson (1981) selittivät skeemojen avulla, miten kognitiivinen taito automatisoituu ja siten suoriutuminen nopeutuu harjoituksen myötä. Anderson *et al.* (1981) esittivät täsmällisiä laskennallisia malleja skeemojen käytöstä geometrian todistustehtävissä, jotka ovat osa matemaattista ongelmanratkaisua. Analogisuus matemaattisen ongelmanratkaisun ja ohjelmointitehtävien ratkaisun välillä on merkittävä. Molemmissa tapauksissa ratkaisu koostuu toisiaan seuraavista askelista, jotka koostuvat formaalisti määritellyistä operaatioista. Anderson *et al.* osoittivat malliensa vastaavuuden ihmisen ajatteluun vertaamalla mallien toimintaa koehenkilöiden haastatteluista kerättyihin ääneenajatteluprotokolliin. He osoittivat, että skeemat ovat hyvä malli ongelmanratkaisussa käytetystä tiedosta. Lisäksi he esittivät mallin siitä, miten skeemoja opitaan esimerkkien ja määritelmien perusteella. Anderson *et al.* esittivät laskennallisen mallin avulla myös esimerkin siitä, miten skeemat voivat rakentua hierarkkisesti siten, että yliskeeman osana on aliskeemoja.

1.3.2. Ohjelmoinnin tyylisäännöt

Joissakin ohjelmoinnin psykologian tutkimuksissa on otettu esiin sellainen ohjelmointitiedon laji kuin *tyylisääntöjen* (rules of discourse) tuntemus (esim. Joni & Soloway, 1986). Esimerkki tällaisestä säännöstä on: ”älä kirjoita ohjelmaan lauseita, jotka eivät vaikuta ohjelman toimintaan mitenkään”. On osoitettu, että tyylisääntöjen noudattaminen tai nou-

dattamatta jättäminen vaikuttaa selvästi siihen, onnistuvatko ohjelmoijat tunnistamaan skeemoja ohjelmakoodista (Soloway & Ehrlich, 1984). Brooksin (1990) mukaan tyylisäännöt ovat interpersonaalista (henkilöiden väliseen vuorovaikutukseen liittyvää) tietoa, sillä niiden tarkoitus on nimenomaan mahdollistaa toisen henkilön kirjoittaman ohjelmakoodin ymmärtäminen. Tyylisäännöt eivät siis varsinaisesti ole skeemoista irrallinen tiedon muoto, koska niiden tarkoitus on tuoda skeemat selvästi esiin koodista. Toisaalta yllä mainittu Solowayn ja Ehrlichin tutkimus myös vahvistaa sen, että ohjelmoijat todella käyttävät skeemoja ohjelmien kognitiivisessa käsittelyssä, koska ilman skeemojen olemassaoloa tyylisäännöilläkään ei olisi merkitystä. Täten tyylisääntöjen itsenäinen asema omana ohjelmointitiedon lajinaan on kyseenalainen.

Ohjelmistotuotannon alalla taas usein nähdään ohjelmakoodia koskevien tyylisääntöjen (tässä asiayhteydessä englanniksi esimerkiksi code conventions tai coding standards) määrittäminen ja noudattaminen osana tehokasta ohjelmistonkehitystä. Tällöin tyylisäännöillä tarkoitetaan eksplisiittisesti ja kirjallisesti annettuja ohjeita siitä, millaista tyyliä ohjelmakoodin tulisi noudattaa. Perustelu tällaisten ohjeiden käytölle ja noudattamiselle esitetään yleensä varsin pragmaattisella tasolla, eli sen tarkoitus on, että ohjelmakoodia olisi helpompi ymmärtää, ilman sen täsmällisempiä psykologisia perusteluja (esimerkiksi Sun Microsystems, 1999).

1.3.3. Sovellusaluetta koskeva tieto

Yksinkertaisia harjoitustehtäviä lukuun ottamatta kaikki tietokoneohjelmat kirjoitetaan siksi, että niitä tarvitaan johonkin todelliseen käyttötarkoitukseen. Tämä todellinen käyttötarkoitus on ohjelmoinnin kannalta *sovellusalue* (*application domain*, usein myös pelkästään *domain*). Voidakseen suunnitella ja toteuttaa hyödyllisen ohjelman, ohjelmoijan on jossain määrin ymmärrettävä sovellusalueen käsitteitä ja sovellusalueen asioiden toimintaa. Esimerkiksi voidakseen tehdä pankkitalletusten käsittelyyn liittyvän ohjelman, ohjelmoijan on ymmärrettävä mitä ovat tili ja korko sekä miten korko lasketaan.

Brooks (1983) hahmotteli teoriaa ohjelman ymmärtämisen prosessista. Hänen mukaansa ohjelman bottom-up-tyylinen ymmärtäminen, jossa ohjelmaa luetaan lause lauseelta ja vähitellen muodostetaan käsitystä kokonaisuudesta, on vain tehokkaampien ymmärtämis-

strategioiden epäonnistuessa käytettävä erikoistapaus. Brooksin mukaan ymmärrettäessä muodostetaan sellaisia kognitiivisia tietoesityksiä, joiden avulla ohjelmoija pystyy muodostamaan yhteyksiä sovellusalueen käsitteistä ohjelman toteutukseen eli ohjelmakoodiin asti. Sovellusalueen käsitteet ja ohjelmakoodi ovat käsitteellisesti niin kaukana toisistaan, että tasojen välille tarvitaan useita välitasoja. Tasoja voivat olla esimerkiksi seuraavat: todellisen maailman oliot, olioihin viittaavat tunnisteet ohjelmassa, olioiden käsittelemiseen tarvittavat tietorakenteet sekä ohjelman lauseet, jotka käsittelevät näitä tietorakenteita. Brooksin mukaan tehokas ohjelman ymmärtäminen tapahtuu siten, että ymmärtäjä muodostaa korkean tason hypoteeseja ohjelmasta ja etsii ohjelmasta skeemojen mukaista evidenssiä hypoteeseilleen. Jos hypoteesille löytyy vahvistusta, hypoteesia tarkennetaan kohti konkreettisemman tason käsitteitä. Jos tukea ei löydy, hypoteesi korvataan uudella. Brooksin mukaan korkean tason hypoteesien muodostamista ohjaa sovellusaluetta koskeva tieto, eli kokemukseen perustuvat käsitykset siitä, mitkä ovat sovellusalueen keskeisiä käsitteitä ja millaisia ovat käsitteiden suhteet toisiinsa.

Adelson ja Soloway (1985) tutkivat ohjelmoijien kognitiivisia prosesseja ja aiemman kokemuksen vaikutusta näihin prosesseihin. Tätä tutkimusta käsitellään tarkemmin luvussa 1.4.1. Adelson ja Soloway kuvasivat ohjelmoijan aiempaa kokemusta kahdella eri ulottuvuudella. Ensimmäinen ulottuvuus oli, onko ohjelmoija työskennellyt vastaavanlaisen ohjelman osan kanssa aiemmin, ja toinen ulottuvuus oli, onko ohjelmoija työskennellyt saman sovellusalueen parissa aiemmin. Kummallakin ulottuvuudella todettiin olevan vaikutusta ohjelmoijan työskentelyyn.

Aiemmin ohjelmointiskeemoja tarkasteltaessa todettiin, että skeemat voivat muodostaa hierarkkisia rakenteita ja sisältää reunaehtoja sille, missä tilanteissa niitä voidaan käyttää. Tässä luvussa mainittujen tutkimusten perusteella käsitystä ohjelmointiskeemoista voidaan tarkentaa niin, että skeemojen muodostamat hierarkkiset rakenteet saattavat liittyä toisaalta ohjelman kuvauksen eri abstraktiotasoihin, mutta toisaalta myös monitasoiseen rakenteeseen, joka kuvaa yhteyksiä ohjelman sovellusalueen ja ohjelman toteutuksen välillä. Nämä hierarkiat saattavat kuitenkin mennä pitkälti lomittain, koska ohjelman kuvauksen ylemmät abstraktiotasot saattavat luontevasti sisältää yhteyksiä myös sovellusalueen korkean abstraktiotason käsitteisiin. Lisäksi osan skeeman käytön reunaehdoista muodostaa tieto siitä, minkä sovellusalueen ohjelmissa kyseisen skeeman toiminta tunnetaan.

1.3.4. Kognitiiviset strategiat ohjelmoinnissa

Ei ole selvää, missä määrin ohjelmoinnissa käytetyt strategiat muodostavat oman, muista riippumattoman tiedon lajin ja missä määrin käytettyjen strategioiden erot johtuvat vain käytettävissä olevan tiedon eroista. On mahdollista, että ohjelmointiskeemojen saatavuus ja ohjelmointiskeemoihin koodattu kontekstuaalinen tieto ohjaavat käytettyjen strategioiden valintaa enemmän kuin skeemoista riippumaton ohjelmointiin liittyvä strateginen tieto. Skeemoista riippumaton strateginen tieto olisi luonteeltaan joko proseduraalista tai heuristista ja luultavasti ohjaisi sitä, millaisia hypoteeseja ongelmanratkaisutilanteessa ryhdytään generoimaan ja testaamaan. Lisäksi strategiat voivat käsittää suunnitteluprosessin hallintaan liittyviä keinoja. Niiden avulla voidaan hallita esimerkiksi sitä, missä järjestyksessä suunnitelman eri osia työstetään.

Ristin (1989) mukaan uudessa ja riittävän vaikeassa tilanteessa ohjelman suunnitteluprosessi etenee siten, että ensin suunnitellaan (luomalla tai valitsemalla) ohjelman pääasiallisen tavoitteen toteuttava koodi. Tätä sanotaan ”fokaaliriviksi” eli ohjelman keskeisimmäksi riviksi. Muu ohjelman rakenne johdetaan tästä fokaalirivistä lähtien tavoitteita takaisinketjuttamalla (goal back-chaining). Takaisinketjutus tarkoittaa, että kutakin löydettyä suunnitelmaa (plan) kohden tunnistetaan sellaiset suunnitelmat, joista löydetty suunnitelma on riippuvainen, ja päätetään toteuttaa myös nämä tunnistetut suunnitelmat. Tämäntyyppistä suunnittelustrategiaa käyttävät Ristin mukaan aloittelijat aina ja ekspertit uusissa tai vaikeissa tilanteissa. Tutuissa tai helpoissa tilanteissa ekspertit pystyvät tuottamaan ohjelman rakenteen top-down-tyylillä, eivätkä tarvitse takaisinketjutusta. Ristin mukaan tämä ero skeemojen käytössä aiheutuu kokemuserosta yksittäisten skeemojen käytössä. Ei siis vaikuta siltä, että tällainen strategian käytössä havaittu ero johtuisi jostakin erityisestä strategisesta taidosta, vaan se on palautettavissa eroihin ohjelmointiskeemojen tuntemuksessa.

Perkins ja Martin (1986) tutkivat erilaisten ongelmatilanteissa saatujen vihjeiden vaikutusta ohjelmoinnin aloittelijoiden suoriutumiseen annetuista ohjelmointitehtävistä. Oppilaan päädyttyä tilanteeseen, josta ei pystynyt omin avuin etenemään, hänelle annettiin järjestyksessä erilaisia vihjeitä. Kussakin ongelmatilanteessa ensin annettiin strategisen tason vihje, joka ei millään tavalla liittynyt käsillä olevaan ongelmaan, esimerkiksi ”Miten ilmaisisit ongelman omin sanoin?” Jos tämä ei auttanut, oppilaalle annettiin tilannekohtainen

tekninen vihje, jonka tarkoitus on ohjata ratkaisua oikeaan suuntaan, esimerkiksi ”Voisiko vika olla for-lauseessa?” Jos tämäkään ei auttanut, oppilaalle kerrottiin ratkaisu hänen ongelmaansa. Eräs tutkimuksen tulos oli, että 33 prosentissa tapauksista pelkän strategisen tason vihjeen antaminen auttoi opiskelijaa eteenpäin. Tästä voidaan päätellä, että paremmat strategiset taidot olisivat jossain määrin auttaneet opiskelijoiden ongelmanratkaisua.

Edellisessä strategiat tarkoittivat ongelmanratkaisustrategioita. Strategiat voivat liittyä myös oman suunnittelutyöskentelyn hallintaan. Esimerkki tästä on tapa, jolla ohjelmoijat Adelsonin ja Solowayn (1985) kokeessa painoivat suunnittelun aikana mieleensä kohtia, joihin tulee palata myöhemmin.

Missään tässä luvussa mainituissa tapauksissa ei ole selvää, että strategiat liittyisivät nimenomaan ohjelmointiin. On mahdollista että strategiat ovat yleisiä kognitiivisen prosessoinnin (esimerkiksi suunnittelun) strategioita, jotka soveltuisivat yhtä hyvin ohjelmoinnin lisäksi myös muihin kognitiivisiin taitoihin.

1.4. Asiantuntijoiden ajattelu

Jos hyväksytään oletus, että mentaalisilla malleilla ja mentaalisilla prosesseilla on joitakin yleisiä ominaisuuksia, voidaan olettaa, että eksperttejä tutkimalla saaduista tuloksista osa on yleistettävissä myös aloittelijoihin. Tässä luvussa esiteltävät tutkimukset kuvaavat sellaisten henkilöiden käyttäytymistä, jotka ovat tässä tutkielmassa tutkittuihin henkilöihin verrattuna erittäin edistyneitä ammattilaisia. Tulokset koskevat mentaalisten mallien ominaisuuksia ja mentaalisen simuloinnin prosessin piirteitä. Näiden ominaisuuksien ja piirteiden on mielekästä olettaa olevan ainakin jossain määrin yleistettävissä myös aloittelijoiden kohtaamiin tilanteisiin, joissa tehtävien muoto (ohjelman toiminnan tarkastelu) on sama, mutta toiminnan kohde on eri (joko laajahkon ohjelmiston osat tarkasteltuna abstraktilla tasolla tai pienen ohjelman yksittäiset lauseet).

1.4.1. Simulointi ohjelmaa suunniteltaessa

Adelson ja Soloway (1985) tutkivat aiemman kokemuksen vaikutusta ohjelmiston suunnitteluun. He erottelivat aiemman kokemuksen kahteen eri ulottuvuuteen, sovellusalueeseen (domain) liittyvään yleiseen kokemukseen ja kokemukseen tietystä suunniteltavasta asiasta. Esimerkkinä sovellusalueesta on yleisesti tietoliikenne, ja esimerkkinä tietystä suunniteltavasta asiasta on sähköpostiohjelmisto. Jos ohjelmoijalla on kokemusta jonkin sovellusalueen ohjelmistojen suunnittelusta, hänellä on hallussaan ohjelmointiskeemoja, joita hän voi jatkossa hyödyntää samankaltaisten asioiden suunnittelussa. Adelsonin ja Solowayn tarkoitus oli selvittää, mitkä osat suunnittelijan osaamisesta riippuvat sovellusaluetta koskevasta tiedosta ja mitkä ovat yleistä sovellusalueesta riippumatonta tietoa.

Adelsonin ja Solowayn tutkimus käsitteli ohjelmistojen suunnittelua. Koehenkilöitä ei siis pyydetty tuottamaan tai lukemaan ohjelmakoodia. Koehenkilöiden tehtävänä oli tuottaa korkean tason suunnitelma järjestelmästä, jonka toiminnan lyhyt kuvaus annettiin tehtävänantona. Koehenkilöt työskentelivät kunkin tehtävän parissa noin kaksi tuntia. Tutkimuksessa oli koehenkilöinä kolme kokenutta ohjelmistosuunnittelun ammattilaista (ekspertit) ja kaksi suunnittelijaa, joilla oli vähemmän kuin kaksi vuotta kokemusta ohjelmistosuunnittelusta (”noviisit”). Eksperteillä oli vähintään kahdeksan vuotta kokemusta ohjelmistojen suunnittelusta. Molemmilla ”noviiseilla” oli useiden vuosien työkokemus ohjelmoijana työskentelyssä, joten he olivat noviiseja lähinnä vain ohjelmistojen suunnittelun suhteen.

Adelsonin ja Solowayn tulosten mukaan vain ekspertit pystyivät simuloimaan ohjelmistojen toimintaa. Noviisien käyttämät representaatiot eivät olleet riittävän dynaamisia, että olisivat mahdollistaneet simuloinnin. Noviisit eivät myöskään pystyneet tarkastelemaan järjestelmän eri toimintojen välisiä vuorovaikutuksia ollenkaan, vaan he tarkastelivat vain yhtä toimintoa kerrallaan. Noviisien ymmärrys annetusta ongelmasta ei myöskään ollut tarkka, vaan saattoi sisältää sellaisia sekaannuksia, jotka olisivat helposti paljastuneet aikaisessa vaiheessa, jos ohjelmistosta muodostettua mallia olisi pystytty simuloimaan.

Ekspertit muodostivat suunnittelemastaan ohjelmistosta kausaalista tietoa sisältävän mentaalisen mallin, joka mahdollisti ohjelmiston toimintojen simuloinnin. Sopivan mentaalisen mallin muodostamisen edellytys oli riittävän tarkasti käsitteellistetty ja rajattu kuvaus rat-

kaistavasta ongelmasta. Jos ongelman kuvaus ei ollut riittävän tarkka, ekspertit keskittyivät kuvaamaan tehtävänantoon sisältyviä implisiittisiä reunaehtoja eksplisiittisesti, kunnes olivat muodostaneet riittävän tarkasti rajatun kuvauksen ongelmasta. Vain riittävän yksityiskohtaista mallia voitiin simuloida. Simulointiin tarvittavan yksityiskohdan puuttuminen suunnitelmasta oli merkki siitä, että yksityiskohta oli suunniteltava. Globaalin mentaalisen mallin luomisen edellytys oli yhtenäinen esitystapa järjestelmän eri toiminnoille. Yhtenäiseen esitystapaan tuli sisältyä esitys järjestelmän tilasta, jota eri toiminnot manipuloivat.

Ekspertit laajensivat suunnitelmiaan systemaattisesti. He aloittivat suunnittelun korkean abstraktiotason suunnitelmasta ja määrittivät vähitellen lisää yksityiskohtia, ja sillä tavoin etenivät kohti konkreettisen toteutuksen tasoa. Ekspertit työstivät suunnitelmia ”leveysjärjestyksessä” eli suunnittelivat samalla yksityiskohtaisuuden tasolla järjestelmän kaikki komponentit ja etenivät vasta sitten seuraavalle yksityiskohtaisuuden tasolle. Samalla yksityiskohtaisuuden tasolla pysyminen oli olennaista, jotta mentaalista mallia pystyttiin simuloimaan. Ekspertin kohdatessa potentiaalisen ongelman, jonka tarkastelemiseksi kehitteillä olevaa suunnitelmaa olisi pitänyt tarkastella toisella yksityiskohtaisuuden tasolla, hän painoi huomion muistiinsa. Tällaista ongelmakohtaa palattiin tarkastelemaan myöhemmin, kun muutenkin oltiin siirtymässä toiselle tarkastelutasolle. Syyksi tällaiselle käytökselle esitettiin, että työmuistissa pidettävää mentaalista mallia ei haluttu häiritä tarkastelutason vaihtamisella. Ekspertit käyttivät nimiä kuvaamaan heille tuttuja skeemoja. Tarvittaessa työstettävän ratkaisun osaksi jotakin ennestään tuttua skeemaa, skeemasta palautettiin mieleen vain nimi, ei yksityiskohtia. Ekspertit luottivat voivansa palauttaa yksityiskohdat mieleensä tarvittaessa.

Jos globaali mentaalinen malli oli olemassa, ohjelmiston osien suunnittelujärjestys vastasi järjestystä, jossa ohjelmiston toimintoja suoritettaisiin ohjelmistoa käytettäessä. Jos globaalia mallia ei ollut käytettävissä, suunnittelujärjestys saattoi rikkoa tätä sääntöä vastaan ja osia saatettiin suunnitella esimerkiksi siinä järjestyksessä, jossa ne oli mainittu tehtävänannossa. Ekspertin joutuessa suunnittelemaan ohjelmistoa itselleen vieraalle sovellusalueelle, hänellä ei enää ollut käytössään suurta joukkoa valmiita skeemoja. Tällöin suunnitelman globaali simulointi ei enää onnistunut. Ekspertit pystyivät tällöinkin simuloimaan ohjelmiston eri osien toimintaa lokaalisti. Nämä lokaalit simuloinnit eivät kuitenkaan paljastaneet suunnitteluvirheitä, jotka johtuivat suunnitelman eri osien välisistä vuorovaikutuksista.

Jos oletetaan, että osa ohjelmoijilta havaituista toiminnoista kuvastaa yleisiä kognitiivisia periaatteita, samankaltaista käyttäytymistä saattaa esiintyä aloittelijoilla, mutta käsiteltävien skeemojen (tai muiden yksiköiden) sisältö on eri. Siinä missä ohjelmistoa suunniteltaessa skeemalla saatetaan esittää laajahkon ohjelmiston osaa, aloittelijalla esitysyksikkönä saattaa olla yksittäinen ohjelmointikielen lause.

1.4.2. Simulointi ohjelmaa luettaessa

Détienne ja Soloway tutkivat, millaisten kognitiivisten mekanismien avulla ohjelmoijat käyttivät ohjelmointiin liittyviä tietojaan (Détienne & Soloway, 1990). He kutsuivat tiedonkäyttömekanismeja strategioiksi. Heillä oli koehenkilöinä 22 ohjelmoinnin asiantuntijaa, eikä ollenkaan aloittelijoita. He käyttivät pääpiirteissään samaa menetelmää ja täsmälleen samaa aineistoa kuin Solowayn ja Ehrlichin tutkimuksessa vuonna 1984 (Soloway & Ehrlich, 1984) käytettiin. Käytettynä tehtävätyyppinä oli ohjelmaan jätetyn tyhjän kohdan täyttäminen mahdollisimman mielekkäällä tavalla. Détienne ja Soloway halusivat tunnistaa ohjelmoijien ääneenajatteluprotokollista ohjelmoijien käyttämät strategiat, sekä saada kustakin strategiasta selville:

- Ehdot, joiden täyttyminen johtaa strategian käyttöön.
- Tavoitteen, johon strategian käytöllä pyritään.
- Strategian piirteet, jotka ilmaistaan strategiaa käyttämällä luodun representaation piirteinä.
- Tiedon, jota strategia käyttää.

He tunnistivat neljä strategiaa, joita kuvataan lyhyesti taulukossa 1-1. Vasemmanpuoleisimmassa sarakkeessa mainitaan strategian nimi ja muissa sarakkeissa yllä mainittuja kysymyksiä vastaavat tiedot kyseiselle strategialle.

Taulukko 1-1. Détiennen ja Solowayn (1990) tunnistamat kognitiiviset strategiat, joiden avulla eksperttiohjelmoijat käyttävät ohjelmointitietoa.

Strategia	**Ehdot**	**Tavoite**	**Tulos (representaatio)**	**Käytetty tieto**
Symbolinen simulointi	Ei ehtoja; käytetään, kun ymmärtämisessä ei ole ongelmia	Tunnistaa ohjelmasta skeemoja	Hierarkkinen kuvaus ohjelman sisältämistä ohjelmointiskeemoista	Tieto ohjelmien roolirakenteesta, valmiit ohjelmointiskeemat
Ohjelmoinnin tyylisääntöjen ja tehtäväalueen periaatteiden mukaan päättely	Kohdataan koodia, jota ei osata selittää millään valmiilla skeemalla	Luoda uusi skeema ja varmistaa skeeman sisäinen koherenssi	Sisäisesti koherentti uusi skeema, joka sisältää tiedot koodin osien välisistä kausaalisuhteista	Tyylisäännöt, tehtäväalueen periaatteet ja yleiset ohjelmointiskeemat
Konkreettinen simulointi	Tuttuja skeemoja on koostettu yhteen tavalla, jonka koherenssista ei olla varmoja	Varmistaa skeemojen ulkoinen koherenssi esim. ei-toivottujen interaktioiden varalta	Skeemojen ulkoisen koherenssin varmistaminen spesifissä tilanteessa	Dynaaminen representaatio, joka kuvaa ohjelman tietovuon (data flow)
Skeemojen reunaehtojen käyttö päättelyssä	Kohdataan koodia, joka ei vastaa odotuksia tai on odotusten kanssa ristiriidassa	Varmistaa skeeman sisäinen koherenssi ja joko muokata skeemaa tai aktivoida vaihtoehtoinen skeema	Skeeman muokkaus tai toisen skeeman aktivointi	Valmiisiin skeemoihin sisältyvät reunaehdot

Détiennen ja Solowayn koeasetelmassa kustakin ohjelmasta oli kaksi versiota: skeemojen mukainen ja skeemojen vastainen. Skeemojen vastainen tarkoittaa heidän mukaansa ohjelmoinnissa käytettyjen ”luonnollisten” tyylisääntöjen vastaista tapaa kirjoittaa ohjelmaa.

Mentaalinen malli voi olla luonteeltaan joko symbolinen tai konkreettinen. Symbolinen tarkoittaa yleistä mallia jonkin ohjelman osan toiminnasta kaikissa tilanteissa. Konkreettinen tarkoittaa mallia ohjelman osan toiminnasta jollakin tietyllä syötteellä. Malli voidaan muodostaa tarvittaessa, esimerkiksi luettaessa ennestään tuntematonta ohjelmaa, mutta muodostamisen jälkeen malli saattaa olla mahdollista palauttaa uudelleen mieleen ilman, että se tarvitsee muodostaa kokonaan uudelleen.

Koska noviiseilla on käytössään vain hyvin vähän valmiita ohjelmointiskeemoja, aivan opiskelun alussa kenties ei yhtäkään, heille lähes kaikki tilanteet ovat skeemojen vastaisia. On odotettavaa, että noviiseilla saattaa olla käytössään varsin erilaisia strategioita kuin mitä Détiennen ja Solowayn tutkimuksessa löydettiin. Vähintään Détiennen & Solowayn esittämät tulokset strategioiden käyttötilanteista ja tehokkuudesta, eli kyvystä ratkaista käsillä oleva ongelma ilman virhettä, olisivat noviiseilla kovin erilaisia. Erityisesti symbolisen simuloinnin strategia, joka nimenomaan nojaa siihen, että koodi on valmiiden skeemojen mukaista, tuskin voi noviiseilla olla ”ensimmäinen” ja ensisijainen strategia.

Détiennen ja Solowayn tutkimuksessa käytettiin vain yhtä tehtävätyyppiä. Siten teoria on validoitu vain yhdentyyppisiä tehtäviä käyttäen. On luonnollista, että koehenkilö sovittaa käyttämänsä strategian käsillä olevaan ongelmanratkaisutehtävään. Erilaiset tehtävät saattavat siten tuoda esiin erilaisia strategioita. Nyt käytetty puuttuvan kohdan täydentäminen saattaa ohjata koehenkilöitä ohjelman oletetun tarkoituksen selvittämiseen. Esimerkiksi virheen etsiminen ohjelmasta saattaisi ohjata erilaisten ymmärtämisstrategioiden käyttöön.

1.5. Aloittelijoiden käyttäytyminen

1.5.1. Ohjelmointiskeemojen luonti

Rist (1989) osoitti, että tutussa tilanteessa ohjelmointitehtävän ratkaisua tuotetaan ”forward, top-down” -järjestyksessä ja uudessa tilanteessa ”backward, bottom-up” -järjestyksessä. Ristin mallissa (Rist, 1986, 1989) kullakin ratkaisuskeemalla on kolme osaa: alustus (initialization), laskenta (calculation) ja tulostus (output). Valmiissa ratkaisussa

osat esiintyvät aina tässä järjestyksessä. Ohjelmoijat eivät kuitenkaan aina kirjoita ratkaisun osia tässä järjestyksessä. Jos käytössä ei ole valmista skeemaa halutun tavoitteen toteuttamiseksi, skeema joudutaan luomaan. Tällöin ensimmäiseksi tuotetaan koodi, joka suoraan toteuttaa tavoitteen. Tämä koodi vastaa skeeman laskentaosaa. Pelkkä laskentaosa ei yleensä riitä toimivan suunnitelman toteuttamiseen. Lisäksi tarvitaan yleensä myös alustus- ja tulostusosat, jotka liittävät laskennan ohjelman kokonaisuuteen. Skeemaa luotaessa nämä tuotetaan koodiin vasta laskentaosan tuottamisen jälkeen. Tästä syystä koodin tuottamisjärjestys on skeemaa luotaessa kääntynyt takaperoiseksi (backward). Jos käytössä on valmis skeema, ratkaisu voidaan tuottaa suoraviivaisesti siinä järjestyksessä, jossa osat esiintyvät ohjelmassa. Tällöin skeeman käyttö on suoraviivaista ja automatisoitunutta (Anderson, 1982, Ristin, 1989, mukaan; automatisoitumisesta myös Anderson et al., 1981 ja Neves & Anderson, 1981).

Ylhäältä alas -suunnittelun edellytys on, että suunnittelija tuntee skeemat, joista abstraktilla tasolla ilmaistu ratkaisu koostuu (Rist, 1989). Jos tällainen skeema puuttuu, joudutaan sellainen ensin luomaan, ennen kuin ylhäältä alas -suunnittelua voidaan jatkaa. Tällöin suunnittelu kohdistuu konkreettiselle tasolle, josta myöhemmin voidaan taas palata ylhäältä alas -suunnitteluun. Skeeman luomisen ajaksi suunnittelujärjestykseksi on vaihtunut alhaalta ylös. Näin siis tuttuja skeemoja käytettäessä ohjelmaa voidaan suunnitella ”forward, top-down” -järjestyksessä ja skeemoja luotaessa suunnittelu tapahtuu ”backward, bottom-up” -järjestyksessä.

Rist on esittänyt mallilleen myös empiiristä tukea (Rist, 1989). Ristin empiirinen tutkimus tehtiin käyttäen koehenkilöinä ensimmäiselle ohjelmointikurssilleen osallistuneita kymmentä vapaaehtoista opiskelijaa. Ristin tutkimus oli pitkittäistutkimus, jossa samoja opiskelijoita tutkittiin kuutena viikkona ohjelmointikurssin aikana. Kullakin kerralla opiskelijoille esitettiin yksi tai kaksi ohjelmointitehtävää paperilla ja opiskelijoita pyydettiin kirjoittamaan tehtävän ratkaisuksi tietokoneohjelma kynällä ja paperilla. Opiskelijoita pyydettiin ajattelemaan ääneen työskentelynsä aikana ja heidän työskentelynsä tallennettiin videonauhalle. Videonauhalta pystyttiin rekonstruoimaan järjestys, jossa opiskelijat tuottivat ohjelman osia. Rist tarkasteli erikseen järjestystä, jossa opiskelijat käsittelivät ratkaisujen osia puheessa ja järjestystä, jossa he tuottivat ohjelmakoodin rivejä paperille. Ristin tarkoitus oli tarkastella skeemojen tuttuuden vaikutusta työskentelyyn ongelmanratkaisun forward-backward- ja top-down - bottom-up -ulottuvuuksilla. Ristin merkitsevä tulos oli,

että jos skeema tunnetaan ennestään, työskentely etenee forward- ja top-down-järjestyksessä ja jos skeemaa ei tunneta vaan se joudutaan luomaan ensimmäistä kertaa, työskentely etenee backward- ja bottom-up-järjestyksessä. Oppimisen vaikutusta Rist pystyi tarkastelemaan vertaamalla toisiinsa tilanteita, joissa opiskelija kohtasi tietyn skeeman ensimmäistä kertaa tilanteeseen, jossa samaa skeemaa tarvittiin myöhemmin uudelleen. Ristin tulos oli, että kertojen välillä oli merkitsevä ero, eli oppiminen oli vaikuttanut järjestykseen, jossa ratkaisun osia työstettiin.

Ristin (1989) tulokset ovat samansuuntaisia kuin Adelsonin ja Solowayn (1985) tulokset siitä, missä järjestyksessä suunnittelijat työstävät suunnitelmiaan. Adelsonin ja Solowayn havainnot olivat kuitenkin selvästi epämuodollisemmin esitettyjä ja heidän käyttämänsä tutkimusmenetelmä ei mahdollistanut esimerkiksi tilastollista tarkastelua luotettavuuden varmistamiseksi. Ristin malli oli hyvin yksityiskohtaisella tasolla esitetty ja Rist pystyi esittämään mallilleen empiiristä tukea, joka oli tilastollisesti merkitsevää.

1.5.2. Aloittelijoiden simulointikäyttäytyminen

Perkins *et al.* (1986) tutkivat ohjelmoinnin alkeita opiskelevien nuorten ohjelmointikäyttäytymistä. Heidän tutkimuksensa menetelmänä oli ala-asteen ja yläasteen koululaisten havainnointi ja haastattelut, jotka liittyivät koululaisten ohjelmoinnin oppimiseen. Ala-asteen oppilailla oli käytössään Logo-ohjelmointikieli ja yläasteen oppilailla Basic-kieli. Tutkimuksen tarkoitus oli tunnistaa ohjelmoinnin alkeiden oppimisen edellytyksiä. Tätä tehdessään tutkijat päätyivät kuvailemaan simulointitaitoa ja sen merkitystä oppimiselle.

Kvalitatiivisia tutkimusmenetelmiä käyttäen he tunnistavat oppilaiden käyttäytymisestä piirteitä, jotka joko edistivät ohjelmoinnin oppimista tai haittasivat sitä. Tällaisia seikkoja olivat oppilaiden käyttäytymismalli ohjelmointitehtäviä ratkaistaessa, ohjelmakoodin tarkka seuraaminen, mahdollisten ratkaisutapojen rakentelu, sekä kyky jakaa ongelmia osiin mielekkäästi. Näistä ohjelmakoodin tarkka seuraaminen (close tracking of code) tarkoittaa käytännössä ohjelman toiminnan simulointia.

Perkins *et al.* tunnistivat oppilaista erilaisia suhtautumisia ongelmanratkaisutilanteeseen. Suhtautumistavan perusteella he kutsuivat joitakin oppilaita jatkajiksi (movers) ja toisia

pysähtyjiksi (stoppers). Kohdatessaan ongelmia jatkajat yrittivät ratkaista ongelmaa jollakin tavalla, kun taas pysähtyjät totesivat, että eivät tiedä mitä pitäisi tehdä, eivätkä osanneet jatkaa ongelman ratkaisua. Tutkijoiden havaintojen perusteella ero johtui affektiivisista tekijöistä, kuten suhtautumisesta virheiden tekemiseen. Pysähtyjät ottivat pientenkin virheiden kohtaamisen raskaasti, turhautuivat ja etäännyttivät itsensä ratkaistavasta ongelmasta. Jatkajat taas pystyivät tunnustamaan virheiden teon ja yrittivät selvittää virheen syytä ja korjata virhettä. Jatkaminen tai pysähtyminen muodostivat jatkumon, jonka toisessa ääripäässä oli ”äärimmäisiä jatkajia”, jotka jatkoivat toimintaansa lukuisilla korjaus- ja muutosyrityksillä pysähtymättä välillä miettimään tai suunnittelemaan toimintaansa. Tutkijoiden mukaan myös tällaiset oppijat etäännyttivät itseään ongelmasta, mutta tekivät sen peittämällä turhautumisen runsaalla ja lakkaamattomalla toiminnalla. Tämän havainnon perusteella siis menestyksekkään ongelmanratkaisun edellytys on sopiva tasapaino ajattelun ja toiminnan välillä.

Perkinsin *et al.* mukaan ohjelmakoodin tarkka seuraaminen on tärkeä taito kaikille ohjelmoijille. Koodin tarkka seuraaminen tarkoittaa ohjelmakoodin lukemista ja sen toiminnan tarkkaa selvittämistä. Jatkossa tästä toiminnasta puhutaan simulointina, vaikka Perkins *et al.* eivät sitä termiä käytäkään. Heidän havaintojensa perusteella simulointi oli hyödyllistä ensinnäkin virheiden havaitsemiseksi kirjoitetusta ohjelmasta jo ennen ohjelman suorittamista ja toiseksi suorituksessa havaittujen virheiden diagnosoimiseksi. Simulointi oli ”mentaalisesti vaativaa” toimintaa, joka edellytti ensinnäkin käytetyn ohjelmointikielen rakenteiden tuntemista. Lisäksi simulointi edellytti *tilakuvauksen* (status representation) ylläpitämistä. Tilakuvaus on dynaaminen, ohjelman suorituksen edetessä muuttuva kuvaus. Tilakuvaus on ongelmakohtainen ja koostuu lähinnä ohjelman muuttujien arvoista sekä ohjelman siihenastisesta tulostuksesta. Periaatteessa ohjelman simulointi olisi täysin mekaanista toimintaa, mutta Perkinsin *et al.* havaintojen mukaan se muodostui usein oppilaille vaikeaksi. (Vaikeutta saattaa selittää luvussa 1.1.1 mainittu Kleinin (1998) tulos kokemuksen merkityksestä mentaalisessa simuloinnissa: kokemusta tarvitaan siihen, että simulointiin osataan valita mukaan oikeat tekijät ja että simulointia osataan yrittää oikealla abstraktiotasolla.)

Ensimmäinen ongelma simuloinnin kanssa oli, että oppilaat eivät aina edes yrittäneet simuloida ohjelman toimintaa, kun se olisi ollut tarpeen. Yrityksen puute saattoi johtua siitä, että oppilaat joko kuvittelivat onnistuvansa ratkaisemaan ongelman muutenkin tai eivät

uskoneet kykyynsä simuloida ohjelman toimintaa. Oppilaan luottamuksen puute omaan simulointikykyynsä saattoi johtua käsityksestä, ettei oppilas tunne ohjelmointikielen toimintaa riittävästi.

Toinen tärkeä tekijä simuloinnin epäonnistumisessa oli usein *omien odotusten projisointi ohjelmaan*. Kuten yleensäkin ihmisen havaintomekanismeissa, odotukset ohjaavat sitä, mitä havaitaan. Jos oppilaalla oli vahvoja käsityksiä siitä, miten ohjelma toimii, hän ei helposti havainnut ristiriitaa sen välillä, miten kuvitteli ohjelman toimivan ja miten ohjelma todella toimi.

Perkins *et al.* eivät ilmaise asiaa niin, mutta heidän käsityksensä ohjelmakoodin ”tarkasta seuraamisesta” eli simuloinnista tarkoittaa selvästi ohjelmaan sisältyvien *kausaalisten suhteiden generointia*. Tämä on yhdenmukaista muiden simuloinnista esitettyjen teorioiden kanssa. De Kleerin ja Brownin simuloitavia mentaalisia malleja käsitelleessä teoriassa eräs mentaalisen mallin muodostamisen vaihe oli kausaalisten suhteiden generointi mallin osien välille. Perkinsin *et al.* maininta simuloinnin tarkoituksesta, eli virheiden löytäminen ja havaittujen virheiden syiden etsiminen, sopivat hyvin siihen, mitä sekä Adelson ja Soloway (1985) että Détienne ja Soloway (1990) esittivät simuloinnin tarkoituksesta. Lisäksi Perkinsin *et al.* käsitys tilakuvauksen ylläpitämisestä on yhdenmukaista sen kanssa, mitä Adelson ja Soloway esittivät tilakuvauksen merkityksestä simuloinnin onnistumiselle.

Lister *et al.* tutkivat useita oppilaitoksia käsittäneessä tutkimuksessa opiskelijoiden taitoja, jotka liittyivät ohjelmakoodin ymmärtämiseen (Lister *et al.*, 2004). Opiskelijoille esitettiin ohjelmien ymmärtämistä testaavia monivalintakysymyksiä. Jälkeenpäin analysoitiin muiden tietojen ohessa myös sitä, millaisia merkintöjä opiskelijat olivat tehneet paperille kysymyksiä käsitellessään. Tarkasteltaessa, millaiset merkinnät esiintyivät useimmin oikeiden tai väärien vastausten yhteydessä havaittiin, että useimmin virheellisen vastauksen yhteydessä esiintyvä ”merkintä” oli tyhjä sivu eli ei merkintöjä lainkaan. Useimmin oikeiden vastausten yhteydessä esiintyivät sellaiset merkinnät, jotka osoittivat opiskelijoiden simuloineen ohjelmien suoritusta. Lister *et al.* eivät tarkastelleet merkintöjen ja oikeiden vastausten suhdetta tilastollisin testein, mutta raportoivat esimerkiksi, että tyhjä sivu esiintyi datassa 256 kertaa ja näistä 50 % oikeiden vastausten yhteydessä. Eräs ohjelman suorituksen seuraamista ilmentänyt merkintätyyppi esiintyi 215 kertaa, joista 75 % oikeiden vastausten yhteydessä. Näistä voidaan laskea testisuure kahden suhteellisen osuuden vertaami-

seksi (Mellin, 1996, s. 347) ja todeta, että ero on erittäin merkitsevä ($p < 0{,}001$). Listerin *et al.* tutkimus siis tukee käsitystä, että ohjelmien simulointi on olennainen osa ohjelmien ymmärtämisen taitoa.

1.5.3. Virhekäsitys: Ohjelma ”ajattelee” kuin ihminen

Pea (1986) hahmotteli teoriaa käsitteellisistä ohjelmointivirheistä (”bugeista”), jotka olisivat käytetystä ohjelmointikielestä riippumattomia. Pean tutkimukset perustuvat Logo-kieltä opiskelleilla 8–12-vuotiailla ja 14–17-vuotiailla oppilailla tehtyihin tutkimuksiin, sekä lukiolaisten Basic-kielen opiskelun havainnointiin.

Pea nimesi kolme käytetystä ohjelmointikielestä riippumatonta käsitteellistä virhettä, joita hän kutsui nimillä parallelismi, intentionaalisuus ja egosentrismi. Hänen hahmottelemansa teorian taustalla oli ajatus, että kaikki nämä kolme virhetyyppiä ovat ilmentymiä yhdestä perustavanlaatuisesta virheestä. Perustavanlaatuinen virhe on, että epäselvissä tilanteissa aloitteleva ohjelmoija olettaa tietokoneella olevan ihmisenkaltaista päättelykykyä, jonkinlaista kokonaisvaltaista hahmotusta ohjelmasta ja sen tarkoituksesta ja että tämän hahmotuskyvyn perusteella tietokone voisi toimia mielekkäästi, vaikka sille annetut ohjelmat olisivat epätäydellisiä.

Pean tutkimusten mukaan aloittelevat ohjelmoijat eivät kysyttäessä tunnusta ajattelevansa näin, mutta kohdatessaan ongelmatilanteita ohjelmien ymmärtämisessä tai tuottamisessa, ilmentävät kuitenkin toiminnallaan tällaista virhekäsitystä. Pean mukaan tämä virhekäsitys on erittäin luonnollinen, koska ainoa käytettävissä oleva hiemankaan tietokoneohjelmoinnin kanssa analoginen kognitiivinen toiminta on toisen ihmisen kanssa kommunikointi, jossa on sopivaa olettaa vastapuolen toimivan mielekkäällä tavalla.

Paralleelinen tarkoittaa rinnakkaista. Parallelismivirheen perusajatus on, että kaikki ohjelman lauseet jollakin tapaa samanaikaisesti, rinnakkain vaikuttaisivat ohjelman suoritukseen. Imperatiivisissa ohjelmointikielissä näin ei koskaan ole, vaan ohjelman toimintaan vaikuttaa vain yksi, parhaillaan käsiteltävä lause kerrallaan. (Javassa ja joissakin muissa ohjelmointikielissä on mahdollista, että ohjelman toiminta koostuu useista keskenään rin-

nakkain suoritettavista ”säikeistä”. Tällaisia ohjelmointikielen piirteitä ei kuitenkaan käsitellä peruskursseilla, joten asialla ei ole tässä yhteydessä merkitystä.)

Pea esitteli parallelismivirheestä kolme vakuuttavaa esimerkkiä. Ensimmäinen esimerkki koskee if-lausetta, joka ei ole toistolauseen sisällä. Pea havaitsi, että useat opiskelijat tulkitsivat tällaisen lauseen ohjelman tilaa koskevaksi pysyväksi ehdoksi, jota tarkkailtaisiin jatkuvasti ja joka vaikuttaisi ohjelman toimintaan koska tahansa, kun if-lauseen ehto-osa tulee todeksi. Pea mainitsi, että eräässä tilanteessa toista vuottaan Basic-ohjelmointia opiskelevista 15 haastatellusta oppilaasta 8 ilmaisi parallelismin mukaisen virhekäsityksen. Toinen esimerkki koskee while-lauseen ehto-osan tarkistamista. Yleinen virhekäsitys oli, että while-lauseen ehto-osan arvoa tarkkailtaisiin jatkuvasti while-lauseen sisällön suorittamisen aikana ja silmukan suorittaminen keskeytettäisiin heti ehdon tullessa epätodeksi. Todellisuudessa ehto tarkistetaan vain yhden kerran kutakin silmukan suorituskertaa kohden ja tarkistus tehdään juuri ennen silmukkaosan suorituksen aloittamista. Pean mukaan kumpikin näistä virhekäsityksistä on yhdenmukainen sen tulkinnan kanssa, joka kyseisillä sanoilla on luonnollisessa kielessä (”if” on suomeksi ”jos” ja ”while” on suomeksi ”sillä aikaa kun” tai ”niin kauan kuin”).

Kolmas esimerkki liittyy sijoituslauseisiin. Pean mukaan jotkut opiskelijat käsittelivät sijoituslausetta ikään kuin se vaikuttaisi sijoituksen kohteena olevan muuttujan arvoon myös sijoituslauseen käsittelemisen jälkeen. Pea käyttää esimerkkinä seuraavaa Basic-kielistä ohjelmaa:

```
AREA = Height * Width
Input Height
Input Width
Print "AREA
```

Ohjelman oikea tulkinta on, että ensimmäisellä rivillä muuttujan AREA arvoksi lasketaan muuttujien `Height` ja `Width` tulo. Koska muuttujille ei sitä lausetta suoritettaessa ole vielä sijoitettu arvoa, käytetään oletusarvona nollaa, joten tuloksi ja siten myös muuttujan AREA arvoksi tulee nolla. Toisella ja kolmannella rivillä luetaan muuttujille `Height` ja `Width` uudet arvot, ja neljännellä rivillä tulostetaan muuttujan AREA arvo. Oikean tulkinnan mukaan neljännellä rivillä siis tulostetaan aina nolla, koska muuttujan AREA arvoa ei muuteta sen jälkeen, kun sille on sijoitettu arvoksi nolla. Pean mukaan useat opiskelijat kuiten-

kin tulkitsivat ohjelman niin, että neljännellä rivillä tulostetaan muuttujien `Height` ja `Width` arvoiksi syötettyjen lukujen tulo.

1.5.4. Olioita koskevat käsitykset

Oliokäsityksistä on julkaistu ainakin kaksi aiempaa tutkimusta (Holland *et al.*, 1997; Eckerdal & Thuné, 2005). Hollandin *et al.* tutkimuksessa opetuskielenä ei ollut Java vaan Smalltalk. Tarkasteltavia opiskelijaryhmiä oli kaksi, ohjelmoinnin alkeita opiskelleet perustutkinto-opiskelijat ja jatko-opiskelijat, joilla oli ohjelmointikokemusta, mutta jotka tutustuivat olio-ohjelmointiin ensimmäistä kertaa. Kumpaakin ryhmää opetettiin etäopetuksena. Hollandin *et al.* tutkimuksessa ei mainittu mitään siitä, millaisella menetelmällä tulokset oli saatu. Voinee vain olettaa, että tulokset perustuivat kirjoittajien käytännöllisiin kokemuksiin.

Holland *et al.* luettelevat mahdollisia väärinkäsityksiä sekä ehdotuksia siitä, miten näitä väärinkäsityksiä voidaan välttää (Holland *et al.*, 1997). Tässä esitetään näistä vain osa. Heidän mukaansa eräs mahdollinen väärinkäsitys on olion ja muuttujan käsitteiden sekoittaminen toisiinsa. Tämä saattaa helposti aiheutua sellaisten esimerkkien käytöstä, joissa oliolla on vain yksi ilmentymämuuttuja. Toinen mahdollinen virhekäsitys on kuvitella, että olion kaikkien ilmentymämuuttujien tulee olla keskenään samaa tyyppiä. Heidän mukaansa ratkaisuna näihin ongelmiin on käyttää esimerkkejä, joissa oliolla on useita keskenään erityyppisiä ilmentymämuuttujia. Yksipuoliset esimerkit saattavat synnyttää mielikuvia esimerkiksi siitä, että oliot ovat pelkästään yksinkertaisia tietovarastoja. Tätä voidaan välttää käyttämällä esimerkkinä oliota, jonka tila vaikuttaa olion käyttäytymiseen. Toinen mahdollinen virhekäsitys on, että olion metodeissa olion tilan muokkaamiseen voidaan käyttää vain sijoituslauseita. Tätä voidaan välttää sillä, että olion tietosisältönä on toisia olioita, jolloin tilan käsittelyyn tarvitaan myös metodikutsuja. Jos esimerkeissä käytetään kerrallaan vain yhtä oliota kustakin luokasta, olion ja luokan käsitteet saattavat sekaantua toisiinsa. Ratkaisuna on aina käsitellä esimerkeissä kerrallaan useaa ilmentymää samasta luokasta.

Eckerdal ja Thuné (2005) tutkivat ensimmäisen ohjelmointikurssinsa juuri päättäneiden opiskelijoiden käsityksiä olioista ja luokista. Menetelmänä heillä oli puoli-strukturoitu haastattelu. Haastatteluihin osallistui 14 opiskelijaa. Haastateltavat oli valittu kurssin

yhteydessä tehdyn kyselyn perusteella siten, että haastateltavat edustaisivat mahdollisimman laajaa kirjoa erilaisia käsityksiä. Eckerdal ja Thuné eivät kertoneet kyselyn sisällöstä tai valintaprosessista tämän tarkemmin. He kuvasivat tekemäänsä analyysiä fenomenografiseksi analyysiksi, joka toteutettiin siten, että kumpikin tutkija analysoi aineiston itsenäisesti ja etsi laadullisesti erilaisia tapoja kuvata olion ja luokan käsitteitä. He mainitsivat, että molempien tutkijoiden tulokset olivat hyvin samanlaisia, mutta eivät kertoneet tarkemmin, miten tämä todettiin.

Eckerdal ja Thuné esittivät tuloksensa kummastakin käsitteestä kolmena kategoriana, joista jälkimmäiset kategoriat sisältävät myös aiempien kategorioiden sisällön. Kategoriat esitetään taulukossa 1-2 samassa muodossa kuin Eckerdal ja Thuné ne esittivät.

Taulukko 1-2. Laadullisesti erilaiset tavat ymmärtää olion ja luokan käsitteet Eckerdalin ja Thunén (2005) mukaan, tämän tutkimuksen kirjoittajan suomentamana.

Olio	**Luokka**
Olio on kappale ohjelmakoodia.	Luokka on kokonaisuus, joka antaa rakennetta ohjelman koodille.
Kuten yllä ja lisäksi olio on ohjelmassa jollakin tavalla aktiivinen osa.	Kuten yllä ja lisäksi luokka on olion ominaisuuksien ja käyttäytymisen kuvaus.
Kuten yllä ja lisäksi olio on malli jostakin todellisen maailman ilmiöstä.	Kuten yllä ja lisäksi luokka on malli jonkin todellisen maailman ilmiön ominaisuuksista ja käyttäytymisestä.

Eckerdal ja Thuné pohtivat tuloksiaan ”variaatioteorian” kautta. Variaatioteorian mukaan jonkin yllä mainitun kategorian mukaisen ymmärryksen muodostamiseksi on tarpeen, että opiskelija havaitsee variaatiota kyseisen asian ymmärtämisen kannalta olennaisella ulottuvuudella. Käytännössä tämä tarkoittaa, että opiskelijoiden tulee kohdata useita esimerkkejä, joiden väliset erot auttavat kehittämään ymmärrystä. Eckerdalin ja Thunén esittämät ulottuvuudet ovat niin yleisellä tasolla, että niistä on vaikea johtaa käytännöllisiä suosituksia, mutta ajatus useista esimerkeistä on yhdenmukainen sen kanssa, mitä Holland *et al.* esittivät (kenties siitäkin syystä, että Eckerdal ja Thuné käyttävät juuri kyseistä Hollandin *et al.* tutkimusta keskeisenä lähteenään).

1.6. Ohjelmointiin liittyvien käsitteiden havainnollistaminen

Koska tässä tutkimuksessa tutkittiin ohjelmoinnin oppimista, on luonnollista pyrkiä tekemään johtopäätöksiä, jotka liittyvät ohjelmoinnin opettamiseen. Mahdolliset johtopäätökset saattaisivat liittyä siihen, miten ohjelmointiin liittyviä käsitteitä tulisi esitellä ja havainnollistaa oppilaille. Jotta tällainen tarkastelu olisi mielekästä, on syytä käsitellä havainnollistamiseen liittyvää aiempaa tutkimusta.

Erilaisten visuaalisten havainnollistusten käyttöä ohjelmoinnin ja laajemmin tietojenkäsittelytieteen opetuksessa on tutkittu 1970-luvun lopulta lähtien (Hundhausen *et al.*, 2002). Kaksi pääsuuntaa tutkimuksessa ovat ohjelmien suorituksen visualisointi (software visualization) ja algoritmien toiminnan visualisointi (algorithm visualization). Algoritmien visualisointi eroaa ohjelmien suorituksen visualisoinnista siten, että siinä ei visualisoida tietyn ohjelmakoodin suoritusta, vaan jonkin algoritmin eli laskentamenetelmän toimintaa käsitteellisellä tasolla. Sama algoritmi voidaan ohjelmoida esimerkiksi eri ohjelmointikielillä, joten algoritmi ei ole suoraan sidoksissa tiettyyn ohjelmakoodiin. Tutkimukset algoritmien visualisoinnista opetuskäytössä ovat antaneet ristiriitaisia tuloksia visualisoinnin hyödyistä. Hundhausen, Douglas ja Stasko (2002) tekivät metatutkimuksen 24:stä kontrolloidusta empiirisestä tutkimuksesta, joissa tutkittiin algoritmien visualisointia opetuskäytössä. Metatutkimuksessa mukana olleista tutkimuksista noin puolessa oli havaittu merkitseviä hyötyjä visualisoinnin käytöstä ja toisessa puolessa ei havaittu hyötyjä. Mukana olleiden tutkimusten piirteitä tarkasteltaessa havaittiin, että jos tutkimuksessa opiskelijat olivat itse saaneet käyttää visualisointityökalua ja tulivat käyttäneeksi visualisoinnin parissa enemmän aikaa kuin kontrollitilanteessa, 71 prosentissa tällaisista tutkimuksista visualisoinnin käytöstä havaittiin merkittäviä hyötyjä, kun taas muissa tutkimuksissa hyötyjä havaittiin vain 33 prosentissa tutkimuksista. Sinänsä ei ole yllättävää, että enemmän aikaa käyttäneet opiskelijat myös oppivat paremmin. Hundhausen, Douglas ja Stasko esittävätkin, että algoritmien visualisoinnin suurin kysymys ei ole se, *mitä* visualisoidaan, vaan *miten* visualisointityökalua käytetään. Heidän mukaansa visualisointityökalun tärkein anti on motivoida opiskelijoita tutkiskelemaan algoritmien toimintaa. On myös esitetty, että algoritmien visualisoinnit näyttävät opettajien mielestä esittävän algoritmin toimintaa selkeästi, koska opettajat jo valmiiksi ymmärtävät algoritmin toiminnan ja siten pystyvät helposti ajattelussaan muodostamaan yhteyden visualisoinnin käyttämien kuvallisten symbolien ja algoritmin toi-

mintaan liittyvän käsitteellisen tiedon välille (Stasko, Badre & Lewis, 1993). Opiskelijoille, jotka eivät ennestään tunne algoritmin toimintaa, tällaisen yhteyden muodostaminen on huomattavasti vaikeampaa. Stasko *et al.* esittävät tämän eron yhtenä syynä sille, miksi monet opettajat kokevat algoritmien visualisoinnin niin hyödyllisenä ja houkuttelevana opetusmenetelmänä.

Saariluoman mukaan tärkeimpiä visualisoinnin haasteita on apperseptio eli mentaalisten representaatioiden muodostaminen yhdistämällä havaintoja ennestään opittuun käsitteelliseen tietoon (Saariluoma, 2001). Tätä mentaalista prosessia ei Saariluoman mukaan tunneta vielä kovin hyvin, mutta eräs sen tunnettu piirre on funktionaalinen järjestyminen eli että kaikilla representaatiossa mukana olevilla osilla ja piirteillä on funktionaalinen syy olla mukana representaatiossa. Visualisointitapoja käsitelevässä taksonomiassaan Roman ja Cox (1993) mainitsevat erääksi visualisointien ominaisuudeksi ”graafisen sanaston” (graphical vocabulary). Tällä tarkoitetaan sitä, että visualisointia muodostettaessa tarvitsee ottaa käyttöön tietyt symbolit, joilla on tietty vastaavuus haluttuihin visualisoinnin kohteisiin eli käsitteelliseen tietoon. Romanin ja Coxin mukaan graafisen sanaston valinta on yleensä mielivaltaista ja graafisten symbolien erilaisilla piirteillä (esimerkiksi koko, muoto ja väri) voidaan monipuolisesti esittää erilaista tietoa. Saariluoman mukaan ajattelua tukevissa esityksissä on tärkeää korostaa pääasioita ja piilottaa tarpeeton tieto. Esitetyistä asioista voidaan johtaa päätelmä, että visualisointeja rakennettaessa on tärkeää valita graafinen sanasto siten, että se esittää sopivan määrän tietoa, ja kommunikoida tähän sanastoon liittyvät merkitykset (esimerkiksi erilaisten muotojen ja värien käsitteelliset merkitykset) katsojille niin, että katsojien on helppo muodostaa yhteys graafisen sanaston ja käsitteellisen tiedon välillä.

Hyvälle opetuskäytössä käytettävälle havainnollistamisvälineelle asetettavia vaatimuksia on myös tutkittu havainnoimalla sitä, miten opettajat ja opiskelijat esittävät ja selittävät ohjelmakoodia laskuharjoitustilanteissa (Lattu, Meisalo & Tarhio, 2001). Tutkimus tehtiin samalla laitoksella ja vastaavalla kurssilla kuin tämä tutkimus, tosin mainittu tutkimus tehtiin syksyllä 1998 ja tämä tutkimus kesällä 2003. Tutkimuksessa havaittiin, että erilaisia strategioita koodin selittämiseen oli useita ja strategiat vaihtelivat lähinnä kurssin vaiheen ja selitettävän koodin monimutkaisuuden mukaan. Selittämisstrategioita voitiin tarkastella lähinnä kahdella eri ulottuvuudella, joista ensimmäinen on selittämisjärjestys ja toinen on käsiteltävien koodin yksiköiden koko. Järjestyksen puolesta koodia havaittiin selitettävän

joko tekstijärjestyksessä eli lukemalla ohjelmakoodia tekstinä alusta loppuun, tai koodia voitiin selittää suoritusjärjestyksessä eli samassa järjestyksessä kuin tietokone suorittaa sitä. Kerrallaan selitettävän koodin yksikön puolesta yleisimpiä strategioita olivat lause kerrallaan selittäminen ja kursorinen selittäminen, jossa koodia selitettiin viittaamatta täsmälleen tiettyihin koodin kohtiin. Muita strategioita olivat ainakin lauseryhmä kerrallaan selittäminen ja metodi kerrallaan selittäminen. Tutkimuksen tärkeimpänä tuloksena oli, että hyvän havainnollistamisvälineen tulee olla riittävän joustava, jotta se tukee erilaisia tapoja selittää koodia. Koska mainitussa tutkimuksessa tutkittiin vain laskuharjoitustilanteita, tulosta ei voida yleistää esimerkiksi itseopiskelutilanteisiin.

Visuaalisesta havainnollistamisesta, eli algoritmien tai ohjelmien suorituksen visualisoinnista, poikkeava tutkimussuunta on tutkia *käsitteellisten mallien* (conceptual models) vaikutusta oppimiseen. Mayer tutki käsitteellisten mallien vaikutusta ohjelmoinnin oppimiseen useissa tutkimuksissa 1970- ja 1980-luvuilla (esimerkiksi Mayer, 1975, 1979, 1981, 1987, Bayman & Mayer, 1983). Mayerin päätelmä oli, että käsitteellisen mallin opettaminen edisti opitun siirtovaikutusta (transfer) uusiin ja haastaviin ongelmiin. Mayerin käyttämät käsitteelliset mallit olivat malleja tietokoneen toiminnasta. Suurimmassa osassa Mayerin tutkimuksista käytetty kieli oli Basic, mutta mainittu päätelmä validoitiin myös toisella imperatiivisella ohjelmointikielellä. Tuloksen yleistettävyyttä olio-ohjelmointikielten oppimiseen ei ole varmistettu, mutta toisaalta tuloksen muoto ei viittaa siihen, että tuloksen pätevyys riippuisi käytetystä kielestä.

Paul ja David Gries (Gries & Gries, 2002) ovat esitelleet tavan havainnollistaa olion ja luokan käsitteitä konkreettisen metaforan kautta, jota voidaan pitää käsitteellisenä mallina olioista ja luokista. Metaforassa luokkaa havainnollistetaan laatikoston yhtenä laatikkona ja oliota laatikostossa olevana arkistokorttina. Viittausta olioon havainnollistetaan mielivaltaisena tunnisteena, joka on kirjoitettu arkistokortin nimikkeeksi ja jonka perusteella kortti voidaan löytää laatikosta. Korttien sisällöksi kirjoitetaan sekä luokan kentät että metodit. Periytymistä voidaan havainnollistaa siten, että kortissa on oma osionsa kullekin peritylle luokalle ja kuhunkin osioon kirjoitetaan juuri siinä luokassa määritetyt kentät ja metodit. Menetelmällä pystytään havainnollistamaan monia olio-ohjelmointiin liittyviä käsitteitä selkeästi ja siten välttämään monia mahdollisia väärinkäsityksiä. Esimerkiksi eräs Hollandin *et al.* (1997) mainitsemista väärinkäsityksistä on samastaa olion tietosisältö olion identiteettiin. Griesien menetelmässä tämä väärinkäsitys vältetään sillä, että olion

identifioi nimikkeeksi kirjoitettu mielivaltainen tunniste (joka vieläpä vastaa likeisesti sitä tosiasiaa, että tietokoneen muistissa olio sijaitsee periaatteessa mielivaltaisessa muistiosoitteessa ja olion identiteetti vastaa olion ilmentymää muistissa). Griesien havainnollistamistapa on erityisen käyttökelpoinen, koska sitä voidaan helposti käyttää yksinkertaisilla havainnollistamisvälineillä. Arkistokortteja voidaan esimerkiksi piirtää taululle tai piirtoheitinkalvolle harjoituksissa ja opiskelijat voivat kopioida piirroksia omiin muistiinpanoihinsa.

Viimeaikaisia ja empiirisesti tutkittuja esimerkkejä ohjelmien suorituksen visualisoinnista alkeisopetusta varten ovat PlanAni (Sajaniemi & Kuittinen, 2005) ja Jeliot (esimerkiksi Levy *et al.*, 2003). Näistä PlanAni havainnollistaa ohjelmointikielten imperatiivisten rakenteiden suoritusta, eikä siten sisällä tapoja olio-ohjelmointiin liittyvien käsitteiden havainnollistamiseen. PlanAni-ohjelman animointi perustuu *muuttujan roolin* käsitteeseen. Muuttujan roolin käsite (Sajaniemi, 2002) perustuu havaintoon, että lähes kaikille aloittelijoiden kohtaamille muuttujille voidaan määrittää selkeä *rooli* ohjelmassa ja että tällaisia rooleja on hyvin rajallinen määrä. Jeliot-ohjelma taas havainnollistaa Java-ohjelmien suoritusta animoimalla ohjelmien suoritusta vaiheittain hyvin yksityiskohtaisella tasolla. Jeliot-ohjelman kuvaesitykset olioista muistuttavat hieman sitä esitystapaa, jota Griesit käyttävät, mutta Jeliot ei havainnollista esimerkiksi periytymistä tai metodien ja olioiden yhteyttä millään tavalla. Toisin kuin Griesien esittämästä havainnollistustavasta, sekä Jeliotin että PlanAnin käytöstä on huolellisia empiirisiä tutkimuksia, joissa havainnollistamisen todettiin ainakin jossakin määrin tukevan abstraktien käsitteiden omaksumista.

Thomas *et al.* (2004) tutkivat oliokaavioiden käyttöä ohjelmoinnin peruskurssilla. Taustalla oli ajatus siitä, että tutkijat, jotka itse opettivat ohjelmointia, käyttivät kaavioita sekä luennoilla että vastatessaan opiskelijoiden kysymyksiin neuvontapäivystystilanteissa. Tutkimuskysymyksinä heillä oli, auttaako oliokaavioiden piirtäminen selviämään paremmin koodin simulointia edellyttävistä kysymyksistä, auttaako osittain piirrettyjen kaavioiden antaminen opiskelijoille heitä suoriutumaan tehtävistä paremmin ja jatkavatko kaavioita saaneet opiskelijat niiden käyttöä jatkossa. Näistä kolmesta kysymyksestä minkään osalta ei saatu merkitseviä tuloksia. Thomasin *et al.* kysymyksenasettelut ovat aiheen tutkimuksen kannalta mielenkiintoisia, mutta voidaan kysyä, vastasiko heidän käyttämänsä interventio tarkoitustaan. Heidän interventionsa oli osittain piirrettyjen oliokaavioiden antaminen opiskelijoille kurssikokeessa, kun kontrolliryhmälle annettiin tyhjät lehtiöt. Olisi mah-

dollista, että radikaalimpi interventio toisi esiin selvempiä tuloksia. Lisäksi tapa, jolla tutkimuksen tulokset on raportoitu, herättää epäilyksiä siitä, että huolellisemmassa tilastollisessa tarkastelussa olisi saattanut tulla esiin asioita, jotka eivät nyt tulleet esiin. Kolmivaiheisessa tutkimuksessa ei verrattu yksittäisten opiskelijoiden tuloksia vaiheesta toiseen, vaan pelkästään suurten ryhmien välisiä eroja kunkin vaiheen sisällä.

1.7. Muita haastattelututkimuksia ohjelmoinnin opiskelijoista

Tässä kerrotaan jo mainituista tutkimuksista sekä muista tutkimuksista esimerkkeinä ohjelmoinnin opiskelijoista käsitelleistä haastattelututkimuksista ja suhteutetaan näitä tutkimuksia tähän tutkimukseen. Jo mainittuja esimerkkejä ohjelmoinnin psykologian tutkimuksista ovat Adelsonin ja Solowayn (1985), Penningtonin (1987) ja Détiennen ja Solowayn (1990) tutkimukset. Nämä kaikki olivat haastattelututkimuksia. Näissä tutkimuksissa haastattelija ei puuttunut haastateltavan työskentelyyn. Mainitut tutkimukset eroavat tästä tutkimuksesta siten, että niissä tutkittiin kokeneita ohjelmoijia ja käytetty menetelmä oli haastattelujen osalta lähellä ääneenajattelumenetelmää. Rist (1989) tutki ohjelmoinnin peruskurssin opiskelijoita useaan kertaan kurssin aikana. Tämän tutkimuksen tavoitteesta ja menetelmästä Ristin tutkimus erosi siten, että siitä validoitiin valmiiksi esitettyä mallia. Opiskelijoiden annettiin työskennellä itsenäisesti (haastattelut alkoivat vasta kurssin kolmannelta viikolta, eivät ensimmäiseltä) ja ääneenajatteluprotokollaa analysoitiin varsin teknisesti.

Esimerkki tutkimuksesta, jossa haastattelija puuttui haastateltavan työskentelyyn, on Perkinsin ja Martinin (1986) tutkimus, jota käsiteltiin jo luvussa 1.3.4. Perkinsin ja Martinin tutkimuksessa haastattelijan puuttuminen haastateltavan toimintaan oli strukturoitua ja asteittain yksityiskohtaisemmiksi käyvien vihjeiden käyttö oli olennainen osa heidän tutkimusasetelmaansa. Perkinsin ja Martinin tutkimus voidaan nähdä välimuotona vapaasti vuorovaikutteisen haastattelun ja haastateltavan itsenäisen toiminnan välillä. Useita esimerkkejä vuorovaikutteisista haastatteluista mentaalisten mallien tutkimuksessa, vaikkakin muutoin kuin ohjelmoinnin yhteydessä, on Gentnerin ja Stevensin toimittamassa klassikkokirjassa *Mental models* (Gentner & Stevens, 1983), esimerkiksi diSessan (1983), Greenon (1983) ja Clementin (1983) tutkimukset.

Aloittelijoiden parissa tehtyjä ohjelmoinnin psykologian tutkimuksia, joissa on käytetty pääasiallisena menetelmänä haastattelumenetelmää, ovat ainakin Eckerdalin ja Berglundin (2005) tutkimus ja Eckerdalin ja Thunén (2005) tutkimus. Näissä tutkimuksissa menetelmänä käytettiin puoli-strukturoitua haastattelua, joka siis muotonsa puolesta oli luultavasti melko lähellä tätä tutkimusta.

Fleuryn (esimerkiksi 1991, 2000 ja 2001) tutkimuksissa käytettiin myös haastattelumenetelmää aloittelijoiden tutkimisessa. Fleuryn käyttämässä menetelmässä olennaisena osana olivat samasta yksinkertaisesta ohjelmakoodista tehdyt muunnelmat, joita opiskelijoiden tuli tutkia ja kommentoida. Kussakin tutkimuksessa oli selkeästi rajattu tutkimusongelma ja kaikki käytetyt koodinäytteet olivat yhden ohjelman muunnelmia. Haastattelun aiheena oli siis pelkästään esitettyjen ohjelmien kommentointi, joten haastatteluissa opiskelijoiden käsityksiä ei tarkasteltu laaja-alaisesti. Toisaalta Fleuryn haastatteluissa tutkittiin nimenomaan opiskelijoiden perusteluita esittämilleen käsityksille, minkä piirteen perusteella Fleuryn tutkimus on samankaltainen tämän tutkimuksen kanssa. Fleuryn tutkimuksissa oli mukana valmiita hypoteeseja ainakin implisiittisesti sen kautta, millaisia muunnelmia tarkasteltavasta ohjelmasta oli otettu tutkimukseen mukaan. Fleuryn tutkimukset olivat siis asetelmaltaan huomattavasti rajatumpia kuin tämä tutkimus.

Myös Fitzgeraldin *et al.* (2005) tutkimuksessa haastattelujen analysointia käytettiin yhtenä tutkimuksen osana. Tutkimus perustui samaan aineistoon kuin Listerin *et al.* (2004) tutkimus, ja tarkoitus oli tunnistaa strategioita, joita opiskelijat käyttivät ohjelmointiin liittyviin monivalintakysymyksiin vastatessaan. Fitzgeraldin *et al.* tutkimuksessa ainoa kuvaus tehdystä analyysistä oli maininta, että käytetty analyysimenetelmä oli ”grounded theory”.

Kaikki edellä mainitut ohjelmoinnin aloittelijoita koskevat tutkimukset Ristin (1989) tutkimusta lukuun ottamatta eroavat tästä tutkimuksesta ensinnäkin siten, että haastatteluille tehtyä analyysiä ei ole esitelty täsmällisesti tai ollenkaan ja toiseksi siten, että niissä opiskelijoita tutkittiin ensimmäisen ohjelmointikurssin lopuksi, ei kurssin aikana. Brucen *et al.* (2004) tutkimuksessa tutkittiin, miten opiskelijat kokevat ohjelmoimaan oppimisen, eli millaisia merkityksiä he antavat ohjelmoinnin oppimiselle. Brucen *et al.* tutkimus on esimerkki aloittelevia ohjelmoinnin opiskelijoita käsittelevästä tutkimuksesta, jossa laadullinen metodologia on täsmällisesti raportoitu. Tutkimus kuitenkin poikkeaa tästä tutkimuk-

sesta ja muista mainituista tutkimuksista oleellisesti siksi, että siinä käsiteltiin subjektiivisia kokemuksia eikä opiskelijoiden tietoja.

1.8. Tutkimuksen motivaatio ja tavoitteet

Tämän tutkimuksen tarkoitus oli selventää yleiskuvaa siitä, millaisia ongelmia ohjelmoinnin alkeita opiskelevilla opiskelijoilla saattaa olla. Yleiskuvaa selvennetään raportoimalla erittäin yksityiskohtaisesti joidenkin opiskelijoiden virheellisiä käsityksiä. Käsitykset raportoidaan kausaalisten mentaalisten mallien tasolla, eli tuloksissa tuodaan esiin, millaisia virheellisiä päätelmiä opiskelijat tekevät ja miten he perustelevat käsityksiään. Näin yksityiskohtaisten tietojen esittäminen on mahdollista ainoastaan yksilöitä tutkimalla. Tällaisten mentaalisia malleja koskevien yksityiskohtaisten kuvausten esittäminen on hyödyllistä ohjelmoinnin psykologian tutkijoille, koska tällainen tieto tulee ottaa huomioon muodostettaessa käyttäytymisestä uusia malleja ja teorioita. Ohjelmoinnin opettajille tieto on hyödyllistä, koska se antaa konkreettisen esimerkin siitä, millaiset väärinkäsitykset oppilailla ovat mahdollisia.

Aiemmassa ohjelmoinnin psykologian tutkimuksessa ei ole esitetty samanlaisia yksityiskohtaisia kuvauksia yksilöiden mentaalisista malleista. Taulukossa 1-3 verrataan tämän tutkimuksen piirteitä joihinkin jo mainittuihin tutkimuksiin. Vertailuun valittiin tutkimuksia, jotka joidenkin piirteiden osalta ovat vertailukelpoisia tämän tutkimuksen kanssa. Vertailu ei ole kattava mukana olevien tutkimusten määrän perusteella, vaan pikemminkin havainnollistava. Kaikkia tutkimuksia ei ole verrattu kaikkien piirteiden osalta, vaan pelkästään niiden piirteiden, joiden osalta vertailu on selvintä. Vertailussa ovat mukana seuraavat tutkimukset: Pea, 1986; Rist, 1989; Fleury, 1991, 2000, 2001; Holland *et al.*, 1997; Lister *et al.*, 2004; Bruce *et al.*, 2004; Eckerdal & Thuné, 2005.

Taulukko 1-3. Joitakin vertailukelpoisia ohjelmoinnin psykologian tutkimuksia verrattuna tähän tutkimukseen.

Piirre	Samankaltaisia	Piirteen suhteen erilaisia
Opiskelijoita tutkittiin kurssin aikana pitkittäisesti ja aivan kurssin alusta lähtien.	Rist, 1989 (tutkittiin kurssin kolmannelta viikolta alkaen)	Fleury, 1991, 2000, 2001 Lister *et al.*, 2004 Eckerdal & Thuné, 2005
Tarkasteltiin olio-ohjelmointi-kielen oppimista.	Holland *et al.*, 1997 Fleury, 2000, 2001 Eckerdal & Thuné, 2005	Pea, 1986 Rist, 1989 Fleury, 1991
Tarkasteltiin yksityiskohtaisia mentaalisia malleja, eli tuotiin esiin myös opiskelijoiden perus-teluja käsityksilleen.	Fleury, 1991, 2000, 2001 Eckerdal & Thuné, 2005	Rist, 1989
Tarkastelua tehtiin tietopohjai-sesti ja ongelmanratkaisun kautta, eikä pelkästään kyselty subjektiivisia vaikutelmia.	Rist, 1989 Fleury, 1991, 2000, 2001 Lister *et al.*, 2004	Bruce *et al.*, 2004 Eckerdal & Thuné, 2005
Tarkasteltiin käsityksiä eksplo-ratiivisesti, eikä ennalta rajattu tarkastelua tiettyyn käsitteeseen tai malliin.	Pea, 1986 Holland *et al.*, 1997	Rist, 1989 Fleury, 1991, 2000, 2001 Eckerdal & Thuné, 2005
Käytetty menetelmä ja analyysi raportoitiin yksityiskohtaisesti.	Rist, 1989 Bruce *et al.*, 2004	Pea, 1986 (ei lainkaan) Holland *et al.*, 1997 (ei lainkaan)

Tässä tutkimuksessa tarkasteltiin opiskelijoita aivan kurssin alusta lähtien. Suurimpia puutteita ohjelmoinnin psykologian tutkimuksessa on, että sellaisten opiskelijoiden mentaalisista malleista, jotka ovat juuri keskeyttämässä kurssin tai juuri keskeyttäneet kurssin, ei ole juuri minkäänlaista tietoa. Tällaista tietoa saadaan vain tutkimalla yksittäisiä kurssilaisia pitkittäisesti kurssin aikana. Tämän tutkimuksen haastateltavista hän, josta käytetään tunnusta A, keskeytti kurssin haastattelusarjan aikana.

Suuri osa ohjelmoinnin psykologian teorioista luotiin 1980-luvulla, jolloin olio-ohjelmointikielet eivät olleet käytössä opetuksessa. Olio-ohjelmoinnin mekanismeihin ja käsit-

teisiin liittyvää viimeaikaista tutkimusta toki on, mutta tutkimus on vielä verrattain vähäistä ja hajanaista. Yksi tämän tutkimuksen tavoitteista oli löytää opiskelijoiden ymmärryksestä sellaisia ongelmakohtia, joita ei ole tutkimuksessa käsitelty aiemmin. Tämän tavoitteen osalta tämän tutkimuksen antina on esimerkiksi tyypin käsitteen problematisointi, jota ei aikaisemmin ole tehty.

Voidaan ajatella, että olio-ohjelmoinnin mekanismit (esimerkiksi perintä ja polymorfismi) ovat tehokkaita abstrahoinnin välineitä ohjelmiston suunnittelijalle, mutta opiskelijalle tämä lisää opittavan tiedon määrää. Olio-ohjelmointikieliin sisältyy käytännössä kaikki samat mekanismit ja rakenteet kuin vanhempiin imperatiivisiin ohjelmointikieliin, joihin oliokielet pohjautuvat. Olio-ohjelmoinnin mekanismit eivät siis ole korvanneet vanhoja mekanismeja, vaan tulleet niiden lisäksi. Tiedon määrään ja moninaisuuteen liittyy ongelmia sekä opiskelijan kannalta että tutkijan kannalta. Opiskelijan kannalta ongelmana voi olla tietomassan jäsentäminen mielekkäästi. Tutkijan kannalta ongelmana voi olla opiskelijoiden havaittujen vaikeuksien jäsentäminen ilmaisuvoimaisten teoreettisten käsitteiden avulla. Ohjelmointikurssilla opittavasta tiedosta osa liittyy siihen, miten opittavat mekanismit toimivat (operationaalinen tieto) ja osa siihen, miten mekanismeja tulisi käyttää (funktionaalinen tieto). Tässä tutkielmassa tarkastellaan operationaalisen ja funktionaalisen tiedon erillään pitämisen merkitystä mentaalisten mallien kannalta. Käsitelty no-function-in-structure-periaate saattaisi tarjota kehyksen toisaalta oppimiseen liittyvien vaikeuksien tutkimiselle ja toisaalta periaatteen, jonka mukaan kurssien oppisisältöjä voitaisiin mahdollisesti selkeyttää.

2. Menetelmä

Tämä tutkimus pyrki dokumentoimaan ohjelmoinnin opiskelijoiden tietojen rakennetta ilman ennakko-oletuksia siitä, millaisia käsityksiä opiskelijoilta saatettaisiin havaita. Tutkimuksen menetelmäksi valittiin puolistrukturoitu haastattelumenetelmä (Hirsjärvi & Hurme, 2000, s. 47). Haastattelumenetelmä valittiin, koska ohjelmoinnin opiskelijoiden mentaalisia malleja on toistaiseksi kartoitettu vain vähän. Haastattelumenetelmä sopii tilanteeseen, jossa halutaan tutkia ennestään vain vähän kartoitettua aihetta (Hirsjärvi & Hurme, 2000, s. 35). Toinen haastattelumenetelmän vahvuus on, että sen avulla on mahdollista syventää

saatavia tietoja. Puolistrukturoidun haastattelumenetelmän joustavuus mahdollisti sen, että haastattelija saattoi syventää haastattelussa esiin tulleiden asioiden käsittelyä lisäkysymyksillä, esimerkiksi kysymällä perusteluja opiskelijoiden esittämille käsityksille. Liitteessä D on yhden haastattelun protokolla esimerkkinä haastattelusta.

Tässä tutkimuksessa raportoidut tulokset ovat enimmäkseen muodoltaan sellaisia, että niissä dokumentoidaan yhden henkilön mentaalinen malli yhdellä ajan hetkellä. Tällöin haastatteluja ei ollut välttämätöntä pitää keskenään vertailukelpoisina. Tässä tutkimuksessa ei tehdä kvantitatiivisia yleistyksiä, vaan tutkimuksen tarkoitus oli tuottaa mahdollisimman syvällistä tietoa yksittäisten henkilöiden mentaalisista malleista.

2.1. Haastattelujen konteksti

Haastattelut järjestettiin ohjelmoinnin opiskelun juuri aloittaneille opiskelijoille toukokesäkuussa 2003, Helsingin yliopiston Tietojenkäsittelytieteen laitoksen järjestämän kurssikokonaisuuden yhteydessä. Kurssit kuuluivat Helsingin yliopiston Avoimen yliopiston opetustarjontaan, joten kursseille sai maksua vastaan osallistua kuka tahansa, koulutustaustasta riippumatta. Helsingin yliopiston perustutkinto-opiskelijoille osallistuminen oli maksutonta.

Kurssikokonaisuus koostui kahdesta kurssista, jotka järjestettiin peräkkäin ilman taukoa kurssien välissä. Kumpikin kurssi päättyi kurssikokeeseen. Yhteensä kurssien laajuus opinnoissa oli viisi opintoviikkoa. Kurssit järjestettiin intensiivikurssina siten, että kurssien keston ajan joka toinen arkipäivä oli kurssin luento ja joka toinen arkipäivä laskuharjoitukset. Kurssikokonaisuus oli suunniteltu siten, että opiskelijoilta odotettiin kokopäiväistä opiskelua kurssien hyväksi. Kurssikokonaisuuden kesto kalenteriajassa oli noin kuusi viikkoa.

Haastattelija toimi kursseilla myös harjoitusryhmien ohjaajana, joten haastattelija oli hyvin perillä kurssien etenemisestä ja sisällöstä. Koska haastattelija toimi kursseilla ohjaajana, hän osallistui myös kurssikokeiden arvosteluun.

Haastattelut pidettiin Tietojenkäsittelytieteen laitoksen tiloissa, tutkimusta varten varatussa neuvotteluhuoneessa. Kunkin haastattelun aika sovittiin henkilökohtaisesti haastateltavan kanssa. Aikaa kutakin haastattelua varten varattiin yksi tunti. Haastattelut äänitettiin äänինauhalle.

2.2. Haastateltavat

Haastateltavat rekrytoitiin Ohjelmoinnin perusteet -kurssille ilmoittautuneista opiskelijoista. Tutkimuksen tekijä kävi kurssin ensimmäisellä luentokerralla kertomassa tutkimuksesta ja pyytämässä kurssilaisilta vapaaehtoisia ilmoittautumisia. Kurssilaiset saivat tutkimuksesta tietoa myös Web-sivulta, joka oli linkitetty kurssin pääsivulle. Kurssilaisia muistutettiin tutkimuksesta uudelleen ensimmäisten harjoitustilaisuuksien yhteydessä, jolloin myös viimeiset ilmoittautumiset otettiin vastaan.

Haastateltavia rekrytoitaessa tutkimukseen osallistumiselle esitettiin seuraavat ehdot: opiskelija oli ilmoittautunut kurssikokonaisuuden molemmille kursseille asianmukaisesti; opiskelija aikoi suorittaa molemmat kurssit; opiskelija aikoi pääsääntöisesti olla läsnä kurssien luennoilla ja harjoituksissa; opiskelija aikoi edetä kurssien opinnoissa kurssien aikataulun mukaisesti; opiskelija ei ollut koskaan ennen opiskellut ohjelmointia; opiskelija pystyi kohtalaisella varmuudella sitoutumaan molempien kurssien ajan kestävään haastattelusarjaan. Haastateltaville ei maksettu tutkimukseen osallistumisesta palkkiota, eivätkä he saaneet tutkimukseen osallistumisesta etua kurssien arvostelussa tai muissa opintosuorituksissa.

Kurssikokonaisuuden jommallekummalle kurssille osallistui 78 henkilöä. Näistä molemmille kursseille osallistui 27 henkilöä. Jommallekummalle kurssille osallistuneista henkilöistä 40 oli naisia ja 38 miehiä. Nuorin osallistuja oli 17 vuotta (syntymävuoden perusteella), vanhin 57 vuotta ja osallistujien keskimääräinen ikä oli 30,1 vuotta. Kurssien osallistujista 72:lla oli ylioppilastutkinto ja 6:lla ei ollut ylioppilastutkintoa. Jommalle kummalle kurssille osallistuneista 41 oli Helsingin yliopiston perustutkinto-opiskelijoita.

Vapaaehtoisia haastateltavia ilmoittautui kuusi henkilöä. Kaikki hyväksyttiin haastateltaviksi siitä huolimatta, että kahdella ilmoittautuneista (haastateltavat C ja D) oli hieman

aiempaa kokemusta ohjelmoinnista. Aiemman ohjelmointikokemuksen määrästä keskusteltiin kunkin haastateltavan ensimmäisellä haastattelukerralla. Keskustelun perusteella haastattelija arvioi aiemman kokemuksen kurssien laajuuteen nähden vähäiseksi. Haastateltavien iät olivat välillä 25–52, keskiarvo 33,8. Haastateltavista viisi oli naisia ja yksi oli mies. Haastateltavien sukupuolijakauma ei vastaa ohjelmoinnin opiskelijoiden sukupuolijakaumaa yleisesti, mutta tällä ei ole merkitystä, koska tässä tutkimuksessa ei pyritä tilastollisiin yleistyksiin. Kaikki haastateltavat olivat Helsingin yliopiston perustutkinto-opiskelijoita. Neljä haastateltavaa oli mukana haastattelusarjassa loppuun asti, eli heiltä saatiin kuusi haastattelua kultakin. Kaksi haastateltavaa keskeytti haastattelusarjan neljännen viikon jälkeen. Heiltä saatiin neljä haastattelukertaa kummaltakin. Yhteensä yksittäisiä haastattelukertoja tutkimuksessa kertyi 32.

2.3. Haastattelujen toteutus

2.3.1. Haastattelusarjan rakenne

Haastattelusarjan rytmitys perustui sen kurssikokonaisuuden rytmitykseen, jota haastateltavat opiskelivat haastattelusarjan aikana. Haastattelusarja koostui kuudesta haastatteluviikosta, jotka vastasivat peräkkäisiä kalenteriviikkoja. Kutakin haastateltavaa haastateltiin kerran haastatteluviikon aikana, eli toistuvasti noin viikon välein. Suhteutettuna kurssikokonaisuuden rytmiin yhden viikon väli haastattelujen välillä tarkoitti pääsääntöisesti, että haastateltava ehti haastattelujen välissä osallistua kahdelle luennolle ja kahteen harjoitustilaisuuteen.

Alla oleva taulukko 2–1 on kalenteriesitys kurssikokonaisuudesta ja haastatteluista. Päivämäärät ovat todellisia päivämääriä vuodelta 2003. Taulukosta nähdään tarkasti haastattelujen sijoittuminen haastatteluviikoille ja haastattelujen ja kurssikokonaisuuksien suhde.

Taulukko 2-1. Kalenteriesitys tutkimukseen liittyvistä haastatteluista sekä kurssikokonaisuuden luennoista, harjoituksista ja kokeista.

Kurssin kesto	Ohjelmoinnin perusteet																							
Päivämäärä	15.5.	16.5.	17.5.	18.5.	19.5.	20.5.	21.5.	22.5.	23.5.	24.5.	25.5.	26.5.	27.5.	28.5.	29.5.	30.5.	31.5.	1.6.	2.6.	3.6.	4.6.	5.6.	6.6.	7.6.
Luento	1				2		3		4				5			6				7		8		
Laskuharjoitus		1				2		3				4		5					6		7			
Kurssikoe																							1	
Haastateltava A						1						2							3					
Haastateltava B						1								2								3		
Haastateltava C								1				2										3		
Haastateltava D						1						2							3					
Haastateltava E								1								2					3			
Haastateltava F								1						2							3			

Kurssin kesto		Java-ohjelmointi																						
Päivämäärä	8.6.	9.6.	10.6.	11.6.	12.6.	13.6.	14.6.	15.6.	16.6.	17.6.	18.6.	19.6.	20.6.	21.6.	22.6.	23.6.	24.6.	25.6.	26.6.	27.6.	28.6.	29.6.	30.6.	1.7.
Luento		1		2		3				4		5					6							
Laskuharjoitus			1		2				3		4					5								
Kurssikoe																		1						
Haastateltava A			4								5								6					
Haastateltava B					4						5													6
Haastateltava C						4					5								6					
Haastateltava D			4							5										6				
Haastateltava E			4																					
Haastateltava F				4																				

Ensimmäisellä rivillä mainitaan palkkeina kurssikokonaisuuden kurssien nimet ja kestot (palkin pituus). Toisella rivillä on mainittu päivämäärät (pystysuuntaan). Seuraavilla riveillä on mainittu kurssien luento-, harjoitus- ja kurssikoekerrat järjestyksessä kurssin alusta saakka. Alimmilla kuudella rivillä on mainittu haastattelupäivät kunkin haastateltavan osalta erikseen. Tummennetulla taustalla on merkitty viikonloput ja viralliset lomapäivät.

2.3.2. Haastattelurungot ja käytetty materiaali

Kunkin viikon haastattelut perustuivat haastattelijan etukäteen valmistamaan haastattelurunkoon. Haastattelurungot ovat liitteessä C. Haastattelurunko koostui avoimista kysymyksistä, käsitteiden määrittelystä, ohjelmakoodin tuottamiseen liittyvistä sanallisista tehtävistä, sekä koodinäytteiden ymmärtämiseen liittyvistä tehtävistä. Kunkin viikon haastattelurunkoon otettiin asioita, jotka juuri sillä viikolla olivat kursseilla ajankohtaisia. Haastattelija saattoi täydentää haastattelurunkoaan viikon kuluessa tai kysyä haastatteluissa kysymyksiä haastattelurungon ulkopuolelta. Haastattelurungon muuttuminen viikon aikana ei haitannut tutkimusta, koska tässä tutkimuksessa ei pyritty haastattelujen väliseen vertailtavuuteen.

Esimerkkejä haastatteluissa käytetyistä **avoimista kysymyksistä** ovat haastateltavan omaan oppimiseen liittyvät kysymykset (esim. ”Mitä olet mielestäsi oppinut edellisen haastattelun jälkeen?”) sekä tietokysymykset (esim. ”Mainitse muutama esimerkki lauseesta, joka luo uuden olion.”). Kuhunkin haastatteluun kuului 5–10 käsitteen määrittely. Käsitteiden määrä ei haastattelujen puolistrukturoidusta muodosta johtuen ollut vakio. Esi-

merkkejä **määriteltävistä käsitteistä** ovat seuraavat: syöte, muuttuja, sijoituslause, olio, luokka ja oliotyyppinen muuttuja. **Ohjelmakoodin tuottamiseen** liittyviä tehtäviä ei kuvata tässä, koska useimmat haastateltavat selviytyivät niistä tehtävistä heikosti, eikä niiden käsittelyä sisälly haastatteluista analysoituihin kohtiin.

Koodinäytteiden lukemista ja niitä koskeviin kysymyksiin vastaamista käsittelevissä tilanteissa haastateltaville annetun tehtävän tarkka muoto vaihteli hieman koodinäytteestä toiseen, mutta tehtävät liittyivät ohjelmien rakenteen ja toiminnan ymmärtämiseen. Kysymykset haastattelija esitti suullisesti. Useimmissa tapauksissa haastateltavaa pyydettiin ensin kertomaan ohjelman rakenteesta ja sen jälkeen kertomaan, miten ohjelman suoritus toimii, kun ohjelma käynnistetään. Ohjelmien suorituksen simulointi tapahtui haastatteluissa aina kynällä ja paperilla. Liitteessä A on ne koodinäytteet, joita käytettiin tätä tutkimusta varten analysoiduissa haastatteluissa. Koodinäytteet oli suunniteltu haastateltavien taitotasoon nähden haastaviksi. Koodinäytteet ja sanalliset tehtävät esitettiin paperilla, jonka haastateltava sai luettavakseen kyseisen tehtävän käsittelyn ajaksi. Tyypillisesti haastateltava teki paperiin paljon merkintöjä, esimerkiksi piti kirjaa muuttujien arvoista. Haastateltavilla oli kaikissa haastatteluissa koko haastattelun keston ajan käytettävissään kynä ja paperia. Haastateltavat saivat halutessaan käyttää kynää ja paperia apuna vastatessaan mihin tahansa kysymykseen, mitä haastattelussa esitettiin. Joissakin tilanteissa haastattelija erityisesti pyysi joidenkin merkintöjen tekemistä. Kaikki haastatteluissa käytössä olleet paperit kerättiin talteen ja niitä käytettiin hyväksi haastatteluja analysoitaessa. Tyypillisesti yhden haastattelun aikana haastateltava teki merkintöjä kahdelle arkille.

2.3.3. Yksittäisen haastattelun kulku

Yhden haastattelun rakenne oli tyypillisesti seuraavanlainen:

1. Aluksi todettiin, milloin edellinen haastattelukerta oli ollut ja millä luennoilla ja missä laskuharjoituksissa opiskelija oli ollut läsnä sitten viime haastattelun. Keskusteltiin myös siitä, paljonko opiskelija oli käyttänyt aikaa kurssin asioiden opiskeluun ja harjoitusten tekemiseen. Lisäksi keskusteltiin siitä, mitä haastateltava koki oppineensa sitten viime haastattelun.
2. Käytiin läpi joidenkin käsitteiden määrittelyjä. Käsitteitä oli tyypillisesti 5–10.

3. Mahdollisesti käsiteltiin joitakin lyhyitä sanallisia tehtäviä, jotka haastattelija luki suullisesti. Esim. ”For-lauseen rakenne ja toiminta”.
4. Käsiteltiin kirjallisesti annettavia koodinäytteitä tai sanallisia tehtäviä. Näitä oli tyypillisesti noin kaksi yhtä haastattelukertaa kohden.

Yhdelle haastattelulle oli tyypillisesti varattu aikaa tasan tunti. Eri haastateltavilta kului saman tehtävän suorittamiseen tai samaan kysymykseen vastaamiseen hyvin erilaisia määriä aikaa. Haastattelija pyrki suunnittelemaan haastattelurungot ja materiaalit niin, että nopeimpienkin haastateltavien kohdalla kysymyksiä ja tehtäviä riittäisi koko haastatteluajaksi, eikä haastatteluaikaa tarvitsisi jättää käyttämättä. Tästä johtuen joidenkin haastateltavien kanssa ehdittiin käsitellä enemmän tehtäviä kuin toisten.

Jos haastateltava halusi spontaanisti puhua jostakin kursseihin liittyvästä asiasta, hänen annettiin näin tehdä. Haastatteluissa ei muodostunut kertaakaan tilannetta, että haastateltavan puhetta jostakin aiheesta olisi tarvinnut keskeyttää.

2.4. Haastattelijan toiminta

Haastattelija toimi pääasiassa haastattelurunkonsa mukaan siten, että haastattelua varten suunniteltuja tehtäviä pyrittiin käymään järjestyksessä läpi niin pitkälle kuin haastattelulle varattua aikaa riitti.

Erilaisia kysymyksiä käsiteltäessä haastattelutilanteen luonne oli erilainen. Suppeita sanallisia kysymyksiä käsiteltäessä tilanne eteni niin, että haastattelija esitti kysymyksen ja haastateltava vastasi. Joidenkin kysymysten kohdalla haastattelija selvitti haastateltavan käsitystä tarkemmin esittämällä tarkentavia kysymyksiä. Haastattelija teki näin tilanteissa, joissa haastateltavan vastaus kuvasti oikeasta ymmärryksestä poikkeavaa mentaalista mallia. Yksittäisen sanallisen kysymyksen käsittely saattoi kestää sekunneista noin minuuttiin.

Koodinäytteiden käsittelemiseen liittyvät tehtävät veivät haastatteluissa aikaa muutamasta minuutista noin neljäänkymmeneen minuuttiin. Haastateltavan simuloidessa koodinäytteen toimintaa haastattelija saattoi kysyä haastateltavalta perusteluja hänen esittämilleen väit-

teille. Jos haastateltava teki selvän virheen, haastattelija korjasi virheen. Jos haastateltava ei ymmärtänyt tai muistanut jotakin asiaa, haastattelija selitti epäselvän asian. Jos haastateltava ei oma-aloitteisesti edennyt tehtävän kanssa, haastattelija ohjasi työskentelyä kysymällä esimerkiksi: ”Mikä on seuraava suoritettava lause?” Simulointitehtäviä yleensä jatkettiin niin pitkään kuin haastateltava pystyi jatkamaan tehtävän käsittelyä tai aika loppui.

Silloin kun haastatteluissa käsiteltiin kirjallista materiaalia, esimerkiksi tulostettuja koodinäytteitä, haastattelija huolehti siitä, että ääninauhalle tallentui selvät tiedot tapahtumien kulusta. Jos haastateltava kirjoitti jotakin paperille tai osoitti jotakin kohtaa paperilla (esimerkiksi ”seuraavaksi suoritus siirtyy *tuohon* kohtaan”), haastattelija luki aina ääneen kirjoitetut asiat tai osoitetun kohdan ohjelmakoodista. Tämä oli hyödyllistä ja tärkeää, koska muutoin tapahtumien kulun selvittäminen jälkikäteen olisi saattanut olla hankalaa.

Haastattelija pyrki toiminnallaan saamaan aikaan mahdollisimman paljon tilanteita, joissa haastateltavat joutuivat esittämään käsityksiä ohjelmissa olevien rakenteiden toiminnasta ja perustelemaan käsityksiään.

3. Analyysi

Tätä tutkielmaa varten pidetyistä 32 haastattelusta litteroitiin ja analysoitiin kokonaan neljä haastattelua ja osia kuudesta muusta haastattelusta. Haastatteluja tehtiin kuudelle haastateltavalle, mutta vain kolmelle haastateltavalle tehtyjä haastatteluja analysoitiin. Analyysi eteni vaiheittain niin, että ensimmäisessä vaiheessa analysoitiin kokonaan neljä haastattelua ja seuraavissa vaiheissa osia muista haastatteluista.

Laadullisen tutkimuksen validiteetin perustaksi on esitetty (Anfara *et al.*, 2002) tutkimuksessa tehtyjen valintojen ja aineiston käsittelyn raportointia riittävän selvästi ja yksityiskohtaisesti, että lukija pystyy esitettyjen tietojen perusteella hyväksymään tai hylkäämään väitteen siitä, että tutkimuksen tulokset ovat luotettavia. Tämän periaatteen mukaisesti tässä luvussa kerrotaan hyvin yksityiskohtaisesti analysoitavan aineiston valinnasta ja analyysiprosessista.

3.1. Analysoitavan aineiston valinta

Tehdyistä haastatteluista litteroitiin ja analysoitiin vain osa. Analysoitavien haastattelujen määrän rajoittaminen on yleistä vastaavankaltaisia haastattelututkimuksia tehtäessä. Esimerkiksi Penningtonin (1987) ymmärtämisstrategioita käsittelevässä tutkimuksessa analysoitiin vain kolme protokollaa. Haastatteluja valittiin analysoitavaksi sillä perusteella, minkä osien analysoinnilla pystyttäisiin parhaiten vastaamaan tutkimuksessa asetettuihin kysymyksiin. Koska suurin osa tehdyistä haastatteluista jätettiin analysoimatta, jokaisen poisjätetyn haastattelun poisjättöä ei perustella erikseen.

Haastatteluja tehtiin yhteensä kuudelle haastateltavalle, mutta vain kolmelle haastateltavalle tehtyjä haastatteluja analysoitiin. Taulukossa 3-1 esitetään analysoitavien haastateltavien valintaan vaikuttaneet tiedot. Taulukossa kaikki kuusi haastateltavaa (tunnukset A:sta F:ään) on karkeasti jaettu kolmeen pariin sen mukaan, miten haastateltavat menestyivät kursseilla. Jakoperusteena on käytetty kurssien kirjanpidosta saatavia tietoja tehtyjen harjoitustehtävien määristä sekä koemenestyksestä. Kustakin jaon mukaisesta parista toisen haastateltavan haastatteluja analysoitiin ja toisen ei.

Taulukko 3-1. Haastateltavien valintaan vaikuttaneet tiedot. Haastateltavat (tunnukset A:sta F:ään) summittaisesti arvioidun menestyksen perusteella.

Menestys	Analyysiin valitut	Analysoimatta jätetyt
Keskeyttänyt	**A**: Keskeytti harjoitusten teon neljännen harjoituskerran jälkeen eli toisella haastatteluviikolla. Osallistui kuitenkin haastatteluihin sarjan loppuun saakka. Ei osallistunut kurssikokeisiin.	**E**: Keskeytti harjoitusten teon jo toisen harjoituskerran jälkeen eli ensimmäisen haastatteluviikon aikana. Keskeytti haastatteluihin osallistumisen neljännen viikon jälkeen. Ei osallistunut kurssikokeisiin.
Kohtalainen	**C**: Osallistui harjoituksiin ja haastatteluihin koko haastattelusarjan ajan. Menestys ensimmäisen kurssin kokeessa heikko, mutta toisen kurssin kokeessa hyvä. Ainoa haastateltu mies.	**F**: Keskeytti sekä kurssille että haastatteluihin osallistumisen neljännen haastatteluviikon jälkeen. Ilmoitti keskeyttämisen syyksi työkiireet. Menestyi kohtalaisesti ensimmäisen kurssin kokeessa.
Erinomainen	**B**: Osallistui sekä kurssille että haastatteluihin koko sarjan ajan. Erinomainen menestys molemmissa kokeissa.	**D**: Osallistui sekä kurssille että haastatteluihin koko sarjan ajan. Koemenestys vielä parempi kuin B:llä. Haastatteluissa selvisi kaikista tehtävistä lähes vaivattomasti.

Taulukossa 3-1 esitetyistä keskeyttäneistä opiskelijoista analyysiin valittiin A, koska hän oli kurssilla mukana pitempään kuin E ja siksi hänen haastattelunsa katsottiin hyödyllisemmiksi kuin E:n. A:n haastatteluja analysoitiin viikoilta 1 ja 3. Koska A keskeytti kurssin harjoitusten teon viikolla 2, A:n viikon 3 haastattelu antaa tietoa sellaisen opiskelijan mentaalisista malleista, jonka opiskelu on juuri joutunut vaikeuksiin. Vaikka A osallistui haastatteluihin loppuun saakka, hänen haastattelujaan viikon 3 jälkeen ei enää analysoitu.

Keskeytysprosentit kursseilla, joilta opiskelijat rekrytoitiin, olivat 31 % ja 24 %. Haastatelluista kuudesta opiskelijasta kolmen keskeyttäminen vaikuttaa paljolta, mutta otoksen koon ja keskeyttämisprosentit huomioon ottaen ei hälyyttävältä. Haastateltujen kanssa käytyjen keskustelujen perusteella kenelläkään syy kurssien keskeyttämiseen ei liittynyt haastatte-

luihin osallistumiseen. Haastateltava A:n osallistuminen kurssille katsottiin keskeytyneeksi, koska A ei itsenäisesti onnistunut ratkaisemaan harjoitustehtäviä viikon 2 jälkeen eikä osallistunut kurssien kokeisiin. Tästä huolimatta A kuitenkin seurasi kurssien luentoja loppuun saakka ja osallistui haastatteluihin. Haastattelusarjan loppuosasta hänen haastattelujaan ei enää analysoitu.

Kohtalaisesti menestyneistä opiskelijoista analyysiin valittiin C, koska haluttiin analysoida viikon 6 simulointiepisodia ja sitä ei keskeyttämisen johdosta ollut F:ltä käytettävissä. Lisäksi C haluttiin valita siksi, että hän oli ainoa haastateltu mies.

Erinomaisesti menestyneistä opiskelijoista valittiin B, koska D osoittautui alustavan aineistoon perehtymisen perusteella osaamistasoltaan niin ylivertaiseksi, että haastatteluissa käytetyt tehtävät eivät juurikaan olleet hänelle haastavia. Hän selviytyi haastattelutehtävistä niin nopeasti ja vaivattomasti, että jotkut suunnitellut haastattelutehtävät ehdittiin esittää vain hänelle, mutta ei muille haastateltaville. Sekä B että D saivat kurssikokeista parhaat mahdolliset arvosanat, mutta koepisteitä tarkasteltaessa D sai paremmat pisteet kuin B. Koska sekä haastattelumenetelmässä että haastattelujen analyysissä kiinnitettiin huomiota normatiivisesti oikeista käsityksistä poikenneisiin käsityksiin, D:n haastatteluista arveltiin alustavan tutustumisen perusteella saatavan hyvin vähän tuloksia.

Tätä tutkimusta varten analysoitiin kokonaan neljä haastattelua ja osia kuudesta muusta haastattelusta. Kokonaan ja osittain analysoidut haastattelut näkyvät taulukossa 3-2. Taulukosta näkyvät haastattelujen sijoittuminen haastateltaville, haastatteluviikoille ja analyysiprosessin vaiheisiin. Osittain analysoiduista haastatteluista analysoitiin viiden haastattelun osalta koodinäytteen simulointia käsittelevä kohta ja yhden haastattelun tapauksessa kohta, jossa haastateltavaa pyydettiin määrittelemään joitakin käsitteitä.

Taulukko 3-2. Analysoidut haastattelut eriteltynä haastateltujen ja haastatteluviikkojen mukaan.

Viikko	Haastateltava A	B	C
1	I: Kokonaan, 42 min	I: Kokonaan, 31 min	
2	II: Simulointi, 20 min	II: Simulointi, 9 min	II: Simulointi, 18 min
3	I: Kokonaan, 47 min	I: Kokonaan, 56 min	
4	(keskeytti kurssin)		III: Käsitteitä, 7 min
5			
6		II: Simulointi, 40 min	II: Simulointi, 26 min

Roomalaisilla numeroilla on merkitty, missä analyysiprosessin vaiheessa haastattelu tai sen osa analysoitiin. Ajat ovat analysoitujen kohtien pituuksia.

Tutkimuksen tekijä tunsi aineiston hyvin, koska oli itse tehnyt haastattelut. Silti ennen kutakin analyysiprosessin vaihetta aineistoa kuunneltiin läpi ääninauhoilta laajemmin kuin mitä vaiheessa lopulta analysoitiin. Kuuntelulla haluttiin varmistaa, että kussakin vaiheessa valitaan analysoitavaksi sellaista aineistoa, joka täyttää vaiheelle asetetut tavoitteet hyvin. Yhteensä analysoitavaksi valittiin aineistoa noin 4 tuntia 56 minuuttia.

3.2. Aineiston analysoinnin eteneminen

Haastatteluaineiston litteroinnissa ja analysoinnissa oli kolme vaihetta. Pääosa analyysistä tehtiin kahdessa ensimmäisessä vaiheessa. Kolmas vaihe sisältää vain yhden haastattelukatkelman analysoinnin, mutta tämä esitetään omana vaiheenaan johdonmukaisuuden vuoksi.

3.2.1. Vaihe I

Haastattelujen analysointi aloitettiin litteroimalla ja analysoimalla kokonaan neljä haastattelua. Neljä haastattelua valittiin siten, että ne olivat kahdelta haastatteluviikolta (viikoilta

1 ja 3) ja kahdelta haastateltavalta (haastateltavilta A ja B), kuten taulukosta 3-2 näkyy. Tällä 2 kertaa 2 -asetelmalla haluttiin selvittää, missä määrin analyysissä voitaisiin hyödyntää pitkittäistä tai poikittaista tarkastelua. Pitkittäisessä tarkastelussa suhteutettaisiin toisiinsa saman haastateltavan toiminnasta tehtyjä havaintoja eri viikoilta. Poikittaisessa tarkastelussa suhteutettaisiin toisiinsa havaintoja eri henkilöistä mutta samalta viikolta. Havaintojen suhteuttamisella toisiinsa tarkoitetaan analyysin *tulosten* suhteuttamista toisiinsa, ei systemaattista *aineiston* piirteiden vertaamista toisiinsa.

Viikkojen 1 ja 3 valinta litteroitaviksi tehtiin sillä perusteella, että viikko 1 antoi hyvän kuvan aloitustasosta ja kahden viikon väli haastattelujen välillä oli kurssin mittakaavassa mielekäs väli edistymisen tarkastelulle. Haastateltavat A ja B valittiin ensimmäisiksi analysoitaviksi, koska molemmat olivat taulukossa 3-1 esitettyjen tietojen perusteella sopivia analysoitavia ja näiden kahden valinnalla saatiin tietoa selvästi eri osaamistasoja edustavista haastateltavista.

Analyysi aloitettiin analysoimalla haastatteluja kokonaan, koska tällä haluttiin selvittää, mistä haastattelujen osista parhaiten voitaisiin saada tutkimuskysymyksen mukaista tietoa, eli havaintoja ohjelmiin liittyvistä mentaalisista malleista.

3.2.2. Vaihe II

Vaiheen I perusteella todettiin, että pitkittäisten ja poikittaisten tarkastelujen hyödyllisyys tämän tutkimuksen tutkimuskysymysten kannalta oli hyvin vähäinen. Analyysiä jatkettiin keskittymällä haastattelujen tiettyyn osaan, koodinäytteiden simulointiin. Niitä haastattelujen kohtia, joissa käsitellään koodinäytteen simulointia, kutsutaan simulointiepisodeiksi. Simulointiepisodit valittiin tarkastelun kohteeksi, koska tutkimuskysymyksenä oli ohjelmia koskevien mentaalisten mallien selvittäminen.

Analysoitaviksi simulointiepisodeiksi valittiin viikkojen 2 ja 6 simulointiepisodit. Viikko 6 valittiin, koska viimeisen analyysiviikon ottaminen mukaan paransi analyysin ajallista kattavuutta ja lisäksi alustavassa kuuntelussa viikon 6 simulointitehtävän käsittely haastatteluissa oli onnistunut. Viikko 2 valittiin, koska sen viikon simulointitehtävä oli haastatte-

luissa onnistunut tutkimuksen kannalta paremmin kuin viikkojen 4 ja 5 tehtävät. Viikkojen 4 ja 5 tehtävät olivat jälkikäteen tarkasteltuna useimmille haastateltaville liian vaikeita.

3.2.3. Vaihe III

Haastateltavan C viikon 6 simulointiepisodia analysoitaessa heräsi kysymyksiä haastateltavan tiettyihin käsitteisiin liittyvistä mentaalisista malleista ja näitä käsityksiä selvitettiin saman haastateltavan viikon 4 haastattelusta, jolloin kyseisten käsitteiden määritelmiä oli kysytty. Tämä muodostaa oman vaiheensa, koska päätös kyseisen kohdan analysoinnista tehtiin vaiheessa II tehdyn analyysin perusteella.

3.3. Analyysin toteutus

Tässä luvussa kuvattu analyysi toteutettiin kussakin edellä kuvatussa vaiheessa sille aineistolle, joka kyseisessä vaiheessa analysoitiin. Haastattelut analysoitiin siten, että ensin haastattelut litteroitiin taulukkolaskentaohjelmaan sanatarkasti. Taulukkolaskentaohjelma oli riittävä, koska tässä tutkimuksessa haastattelujen analysointi ei perustunut tekstisegmenttien luokitteluun. Litteroitujen lausumien yhteyteen merkittiin aikaleimaksi nauhalaskurin numero lausuman alun kohdalla.

Kun haastattelut oli litteroitu, haastatteluja kommentoitiin yksittäisten lausumien kohdille samaan laskentataulukkoon. Kommenteilla lisättiin tulkintaa protokollan puhekieliseen ilmaisutapaan, otettiin kantaa haastateltavien lausumien oikeellisuuten sekä kirjattiin muistiin haastattelun sisäisiä ristiviittauksia. Ristiviittaukset olivat samaan asiaan tai aiheeseen liittyviä lausumia, jotka esiintyivät eri kohdissa haastattelua. Lisäksi kommentteihin kirjoitettiin puhtaaksi haastateltavien paperille kirjoittamia asioita, jotta niitä olisi helppo tarkastella samassa asiayhteydessä kuin puhuttuja asioita. Liitteessä D on yhden haastattelun protokolla esimerkkinä haastattelusta. Protokollaesimerkissä ei ole mukana kommentteja, koska ne olisivat vaatineet taulukkomaisen esityksen ja siten suuren sivumäärän.

Kommentointivaiheen jälkeen kunkin viikon haastatteluille muodostettiin yhteinen analyysipohja. Kokonaan analysoituja haastatteluja varten tehdyt analyysipohjat olivat laajempia kuin pelkkiä simulointiepisodeja varten tehdyt pohjat. Analyysipohjat, samoin kuin haastattelurungot, sisälsivät aiheita, jotka kyseisellä viikolla olivat kurssin kannalta ajankohtaisia. Mukaan otettiin joitakin sellaisia teemoja, jotka eivät suoraan esiinny kurssien oppisisällössä, vaan jotka otettiin mukaan muilla perusteilla. Muita perusteita olivat esimerkiksi teeman psykologinen kiinnostavuus (allaolevasta rungosta kohdat 9 ja 14) tai haastattelijan alustavat havainnot aineistosta (allaolevan rungon kohta 12). Seuraavassa on esimerkkinä viikon 3 analyysipohja:

1. Terminologia
    1.1. Olio
    1.2. Luokka
    1.3. Tyyppi, tyypin käsite
    1.4. Palautusarvo, arvon palauttaminen
    1.5. Kapselointi
    1.6. Algoritmin tai olion tila
    1.7. Muut
2. Syntaksi
    2.1. Erotinmerkit
    2.2. New-lause
    2.3. Kokonaiset lauseet (itse tuottaminen)
    2.4. Metodin parametrit
3. Muuttujat
4. Ohjelman rakenne
5. Suoritusjärjestys
6. Parametrinvälitys
7. Metodien määreet
8. Tehtävien tekeminen, virheiden käsittely
9. Ohjelman tuottamisen prosessi (tehtävä 2)
10. Konstruktori
11. Olioiden viitesemantiikka
12. Pääohjelma itsenäisenä käsitteenä
13. Taulukot
14. Päätelmien teko
15. Muuta
    15.1. This-viittaus
    15.2. Metodien toiminnan ymmärtäminen

Analyysipohjan muodostamisen jälkeen viikon haastattelut käytiin huolellisesti läpi ja poimittiin haastatteluista analyysipohjan kohtiin liittyvät haastattelun kohdat. Kustakin yksittäisestä haastattelusta muodostettiin analyysidokumentti, jonka sisällysluettelona oli analyysipohjan kohdat, ja kunkin otsikon alle koottiin teemaan liittyvät kohdat haastattelusta. Analyysidokumentteihin kirjoitettiin mukaan tulkintaa ja argumentointia siitä, mitä haas-

tattelusta poimitut havainnot ilmentävät. Analyysidokumentteja kertyi yhteensä yli 40 sivua.

Tässä tutkimuksessa käytetty analyysipohja eroaa laadullisissa analyyseissä usein käytetyistä luokittelurungoista. Luokittelurungon muodostaminen on keskeinen osa prosessia, jossa haetaan sopivia käsitteitä tutkittavan ilmiön kuvaamiseksi (Hirsjärvi & Hurme, 2000, s. 147). Tässä tutkimuksessa analyysipohjassa mukana olevat käsitteet saatiin suurimmalta osalta valmiina ohjelmointikieltä koskevasta tiedosta tai ohjelmoinnin psykologian tutkimuksen kysymyksenasetteluista. Analyysipohja oli siis pikemminkin tarkistuslista siitä, mihin teemoihin liittyviä havaintoja haastatteluista *voisi löytyä* kuin luokittelu siitä, mitä asioita haastatteluista *löytyy*. Analyysipohjan merkitys tulosten raportoinnissa on myös vähäisempi kuin vastaavasti luokittelurungon. Jotta analyysidokumenttiin kertyneistä havainnoista päästiin raportoitaviin tuloksiin, piti havaintojen välille löytää loogisia yhteyksiä, jotka sitoivat havainnot johdonmukaiseksi kuvaukseksi haastateltavan mentaalisesta mallista. Siten siis pelkästään havaintojen kuuluminen samaan luokkaan ei olisi tämän tutkimuksen tutkimusongelman mukainen tulos.

Kun kaikki kolme analyysin vaihetta oli toteutettu edellisissä luvuissa kuvatulla tavalla, analyysidokumenttien pohjalta edettiin kohti sitä muotoa, jossa tulokset nyt raportoidaan tämän tutkielman seuraavassa luvussa. Raportoitavaksi valittiin havaintokokonaisuuksia, joissa useampi kuin yksi havainto muodostavat johdonmukaisen kokonaisuuden ja kokonaisuudelle pystytään antamaan tulkinta. Haastatteluista kirjattiin kommentointi- ja analysointivaiheessa suuri määrä yksittäisiä havaintoja, jotka jätetään raportoimatta, koska niille ei pystytty antamaan sellaista tulkintaa tai selitystä, joka nostaisi niitä muuksi kuin triviaaleiksi yksityiskohdiksi. Raportoitavat tulokset syntyivät sellaisen kirjoitusprosessin tuloksena, jossa myös prosessin aikana tarkistettiin havaintoja suoraan haastatteluista. Kutakin raportoitua havaintokokonaisuutta varten käytiin uudelleen läpi ne litteroidut haastattelujen kohdat, joita kokonaisuuden raportoinnissa käytetään. Tällöin varmistettiin haastatteluista, että kaikki esitettyyn kokonaisuuteen liittyvät havainnot oli otettu huomioon.

Havaintokokonaisuuksina esitettyjen tulosten rakenne on sellainen, että niissä useita yksittäisiä havaintoja yhdistetään loogiseksi kokonaisuudeksi. Havaintojen liittyminen samaan kokonaisuuteen selviää monessa tapauksessa vasta siitä, mihin abstraktiin asiakokonaisuuteen liittyvää käsitystä havainnot kuvastavat. Tällaisten yhteyksien huomaaminen ei onnis-

tuisi pelkästään luokittelemalla, vaan edellyttää monivaiheista sykliä, jossa yhdelle havainnolle annettu tulkinta herättää kysymyksen, johon etsitään vastausta sovittamalla samankaltaista tulkintaa muille havainnoille.

Esimerkiksi ilman haastattelujen huolellista analysointia ei olisi selvää, että haastateltavan maininta ”olio on itsessään muuttuja” liittyisi samaan kokonaisuuteen kuin hänen mainintansa ”esimerkiksi ... jos halutaan, että se saa vaikka arvot yksi viiva kaksitoista” muuttujan arvojen rajaamisesta. Analyysissä kuitenkin voitiin todeta, että käsitys ”olio on itsessään muuttuja” saattaa johtua kurssimateriaalissa käsitellystä yksinkertaisesta esimerkistä. Jälkimmäinen maininta taas oli todiste siitä, että haastateltava todella käytti kyseistä esimerkkiä ajattelussaan, koska myös kurssimateriaalin esimerkissä esiintyi täsmälleen sama arvoväli.

Vaikka systemaattinen tekstin segmentointi ja luokittelu olisi saattanut toimia apuna tämän prosessin alkuvaiheessa (joka nyt tehtiin kommentoimalla protokollaa samaan laskentataulukkoon), havainnoista tehtyihin tulkintoihin ja siten raportoituihin tuloksiin sillä tuskin olisi ollut vaikutusta.

4. Tulokset

Tulokset esitellään pääosin sellaisessa muodossa, että esitettyjen väitteiden tai havaintojen yhteydessä olevat lainaukset haastatteluista itsessään perustelevat esitetyt väitteet. Useissa tämän luvun kohdissa viitataan haastatteluissa käytettyihin koodinäytteisiin. Koodinäytteet löytyvät liitteestä A. Lainauksissa esiintyvät kirjaimet A, B ja C tarkoittavat haastateltavia ja H tarkoittaa haastattelijaa. Lainausten yhteydessä esiintyvät luvut tarkoittavat nauhalaskurin kohtaa, jonka kohdalla kyseinen lausuma haastattelussa esiintyy. Yksi nauhalaskurin yksikkö vastaa 3,6 sekuntia.

4.1. Yleiskuva haastateltavien taitotasosta

Yleiskuva haastateltavien taitotasosta muodostaa pohjan muiden tulosten tulkinnalle ja tarkastelulle. Yleiskuvan yhteydessä esitetään tietoja myös siitä, mitä asioita kurssilla käsiteltiin kyseisellä viikolla ja millaisia tehtäviä haastatteluissa käytettiin kyseisellä viikolla. Kukin esitetty havainto liittyy yhteen haastateltavaan ja yhteen tiettyyn haastatteluun. Yleiskuvan hahmottamisessa on muistettava haastattelusarjan ajallinen luonne: on luonnollista, että haastateltavien taitotaso paranee haastattelusarjan ja siten heidän opiskelunsa edetessä.

4.1.1. Viikko 1

Ensimmäisen viikon haastatteluista analysoitiin haastateltavien A ja B haastattelut kokokaan.

Ensimmäisen viikon luennoissa kurssilla on käsitelty muuttujien ja sijoituslauseiden käyttöä, aritmeettisia laskutoimituksia, merkkijono-operaatioita, syöttöä ja tulostusta, vertailuoperaatioita, valinta- ja toistolauseita sekä aloitettu metodien käsittelyä. Tällä viikolla haastattelussa käsitellyssä koodinäytteessä esiintyi muuttujia, sijoituslauseita, for-lause, if-then-else-lause, numeerisia operaatioita ja merkkijono-operaatioita sekä tulostusta.

Haastateltava A hallitsi kurssilla äskettäin käsitellyn terminologian vain osittain. A osasi määritellä käsitteet tietokoneohjelma, algoritmi ja vertailuoperaatio, mutta termi *lause* oli haastateltavalle epäselvä. Haastateltava ilmaisi käsityksen, että ohjelmassa on lauseiden välissä jotakin muuta, joka ei ole lauseita:

> (#155)
> Haastattelija: Mistä tietokoneohjelma koostuu?
> A: — — Siis muuttujat, jotka on lauseessa, se tarvitsee ne lauseet. Ja sitte niitten välillä jotakin toimintoja, elikkä jos... Siis mitä näitten lauseiden välillä tapahtuu. — — Sitte tarvitaan näitä lauseiden välissä olevia... Elikkä siis... Jos, tai ne, lauseita, tai mitä näitä nyt olikaan.

Lauseella haastateltava mahdollisesti tarkoitti tulostettavia asioita. Haastattelijan pyytäessä nimeämään ohjelmasta mahdollisimman paljon eri asioita, haastateltava lausui tulostuslauseen kohdalla seuraavasti:

> (#324)
> A: Sitten on et tulostaa lauseen.

Kielen rakenteiden hahmottaminen ohjelmakoodista ei aina onnistunut. Haastateltava esimerkiksi virheellisesti ymmärsi tulostuslauseessa olevat muuttujien nimet merkkijonoliteraaleiksi. Ohjelman suoritusjärjestyksestä A osasi sanoa, mistä ohjelman suoritus alkaa, mutta kaikissa tilanteissa hän ei osannut vastata kysymykseen, miten ohjelman suoritus etenee seuraavaksi. Haastattelijan ehdotettua, että seuraavaksi vuorossa olisi tulostuslause, A lausui:

> (#516)
> A: Ehkä mä en oo vielä ymmärtäny tätä tulostuslausetta, elikkä mä oon ajatellu että tulostuslause on vaan sille käyttäjälle. Et se ei oo se millä se tietokone operoi, vaan... Mutta mä oon sit ymmärtäny väärin.

A:n käsitys *muuttujan tyypistä* oli hatara, A näki sen ensinnäkin jotenkin ehdollisena (”jos se on määritelty”) ja toisekseen sellaisena, että se voi muuttua. Luvussa 4.2.1 käsitellään tarkemmin A:n käsitystä muuttujan tyypistä.

Haastateltava ei oma-aloitteisesti yrittänyt hahmottaa ohjelman rakennetta aaltosulkeiden mukaan ja pyydettäessä käsittelemään aaltosulkeiden merkitystä, A tulkitsi ne väärin: haastattelijan pyytäessä A:ta merkitsemään paperilla olevaan koodilistaukseen (liite A1), mitkä aaltosulkeet muodostavat parit, A merkitsi parit väärin, järjestykseen 123123, kun oikea olisi 123321. Yrittäessään hahmottaa suoritusjärjestystä ohjelman rakenteen perusteella, A virheellisesti rinnasti aaltosulkeiden merkityksen siihen, miten matematiikassa ilmaistaan laskujärjestystä sulkeilla.

A osoitti hyvää käsitystä siitä, miten ohjelman suoritusta periaatteessa voisi simuloida, sillä hän onnistui kirjaamaan muuttujien arvoja paperille varsin käytännöllisesti. Hän ei kuitenkaan koodin perusteella hahmottanut ohjelman lohkorakennetta, eikä tuntenut for-lauseen osien suoritusjärjestystä, joten hän ei lainkaan pystynyt simuloimaan ohjelman suoritusta.

Viikolla 1 haastateltava B tunsi kysytyt termit muutoin hyvin, mutta termi *sijoituslause* oli hänelle täysin tuntematon. B oli ymmärtänyt *syötteen* käsitteen väärin. Hänen mukaansa syöte oli käännetty mukaan ohjelmaan ja se oli ainoa syöte, mitä hän tähän mennessä oli kurssilla kohdannut. B hahmotti koodinäytteen (liite A1) syntaksin muutoin hyvin, mutta tyhjän merkkijonoliteraalin hän hahmotti väärin (mikä näkyy myös alla olevissa lainauksissa) ja aaltosulkeiden merkitystä hän ei aluksi muistanut. Haastattelijan vihjeen jälkeen B osasi käsitellä aaltosulkeet ja siten ohjelman rakenteen oikein.

Muuttujien arvojen käsittely ja päivitys sujui B:ltä ongelmitta siinä suhteessa, että hän ymmärsi muuttujien toiminnan ja niiden käsittelyyn liittyvän syntaksin. B ei kuitenkaan onnistunut simuloimaan koodinäytteen suoritusta oikein, koska hän ei muistanut tai tuntenut for-lauseen toimintaa tarkasti. Hän tunnisti tai osasi päätellä for-lauseen halutun toiminnan eli *funktion:*

> (#572)
> H: Tota, jos mietittäs niin päin, et mitä tältä ohjelmalta luultavasti haluttais? Eli miten sen i:n arvon haluttais käyttäytyvän?
> B: Musta näyttää siltä, että halutaan, että se kasvaa, kunnes se on yli yhdeksän, jolloin tää for-lause ei enää pyöri, niin sanotusti.

B:n käsitys for-lauseen osien suoritusjärjestyksestä aiheutti ristiriidan halutun toiminnan kanssa. Hän huomasi ristiriidan itse, mutta ei onnistunut korjaamaan käsitystään niin, että olisi pystynyt simuloimaan ohjelman toimintaa. Ristiriita johtui B:n käsityksestä, että samalle riville kirjoitetut lauseet käsitellään vasemmalta oikealle. Hän yritti soveltaa tätä käsitystä for-lauseen sulkeiden sisään kirjoitettuihin lauseisiin, jotka syntaktisesti vastaavat muualla ohjelmassa olevia lauseita, mutta niillä on kuitenkin for-lauseen osina omat erityismerkityksensä. Syntaktinen vastaavuus muualla ohjelmassa oleviin lauseisiin tulee siitä, että for-lauseen osat erotetaan toisistaan puolipisteellä ja muutoinkin ohjelmassa käytetään puolipistettä lauseiden päättymisen merkkinä.

B osasi hahmottaa koko if-else-lauseen funktion oma-aloitteisesti ja myöhemmin simuloinnissaan käytti tätä hahmotusta ilman, että olisi erikseen joutunut käsittelemään lauseen osien merkitykset. Tässä B hahmottaa if-else-lauseen funktion:

> B (403): No sit tulee tää if-lause, jossa tää iso-muuttuja, sen, sen tota niin... Siis kahdella... Tai siis, no... Täähän tarkottaa sitä et sen pitää olla parillinen. Eli jos se on parillinen, niin

> sitten merkin arvo... merkki saa arvon lainausmerkki tähti lainausmerkki, muuten se saa arvon kaksi lainausmerkkiä.

Myöhemmin haastattelussa, noin kuusi minuuttia edellisen kohdan jälkeen, B käyttää päättelyssään sujuvasti hyväkseen edellä esittämäänsä hahmotusta if-else-lauseen funktiosta:

> B (505): Tässä tapauksessa, koska toi ison arvo ei ole parillinen, niin sit se tulostaa, heh, kaks lainausmerkkiä.

4.1.2. Viikko 2

Toisella haastatteluviikolla kurssin asiasisällössä on käsitelty metodien toiminta, olioita ja luokkia sekä verrattu toisiinsa String-olioita ja char-arvoja, jolloin on painotettu eroa sen välillä, että String-oliot ovat olioita ja char-arvot alkeistyyppisiä arvoja. Tämän viikon haastatteluissa simuloitiin koodiesimerkkiä, jossa esiintyi kokonaislukumuuttujia, kaksi sisäkkäistä for-lausetta ja tulostuslause. Esimerkin ulompi for-lause oli tehty ”epätyypilliseksi” for-lauseeksi siten, että lauseen alustusosassa alustettua muuttujaa ei käytetty mihinkään ja lauseen etenemisosassa käytettiin yhteenlaskuoperaation sijasta jakolaskua. Ulomman for-lauseen määrittely oli tällainen:

```
for (int l = 0; luku > 2; luku /= 2) {
```

Simuloitavassa koodinäytteessä käytettiin tarkoituksella ”epätyypillistä” koodia, jotta opiskelijat joutuisivat yksityiskohtaisesti simuloimaan ohjelman toimintaa. Kovin tyypillisten ohjelmien tarkastelussa olisi ollut se ongelma, että opiskelijat pystyisivät mahdollisesti päättelemään ohjelman suoritusta näkemiensä esimerkkien perusteella ilman, että yksityiskohtaisesti simuloisivat koodin toimintaa.

Viikolta 2 analysoitiin kolmen haastateltavan haastatteluista osat, jotka liittyivät koodinäytteen (liite A2) toiminnan simulointiin. Haastateltavat olivat A, B ja C. Kaikille kolmelle haastateltavalle sanottiin koodinäytteen esittämisen yhteydessä, että ohjelman suoritusta koskevien päätelmien tekeminen olisi suotavaa sen asemesta, että simuloi lauseiden jokaisen suorituskerran läpi.

Yhteistä kaikkien kolmen suoriutumiselle tehtävässä oli, että jokainen tarvitsi jonkin verran apua haastattelijalta ulomman for-lauseen yhden tai kahden ensimmäisen iteraatioker-

ran simulointiin. Selvittyään ensimmäisistä iteraatiokerroista jokainen haastateltava pystyi tehokkaasti ja vaivattomasti simuloimaan jäljellä olevat ulomman for-silmukan iteraatiot. Tehokkuudesta oli merkkinä, että haastateltavat pystyivät käsittelemään ulomman for-lauseen iteraatiokerran kaikki vaikutukset yhtenä kokonaisuutena ilman, että olisivat simuloineet sisemmän for-silmukan yksittäisiä iteraatioita. Tämän havainnon lisäksi C:n haastattelusta tunnistettiin selkeä kognitiivinen strategia suorituksen simuloinnille. Strategiaa käsitellään luvussa 4.3.3. Seuraavassa esitetään simuloinnin loppuvaihe kultakin kolmelta haastateltavalta. Lainauksista näkyy, miten kukin haastateltava käsittelee sisemmän for-lauseen suoritusta yhtenä kokonaisuutena, simuloimatta läpi yksittäisiä iteraatioita. Muita haastateltavien simuloinnin yksittäisiä piirteitä ei käsitellä tässä.

Haastateltava A:

A (911): Niin, ja palataan tonne ylempään, jossa me ollaan jääty että luvun arvo on 32.
H (913): Joo.
A (913): Joka, sit, tässä taas sit tehdään niin että jaetaan se kahdella, elikkä me saadaan 16.
H (914): Joo.
A (914): Joka on tän luvun uus arvo.
A (917): Jonka jälkeen tutkitaan että onko se suurempi kun kakkonen, on.
H (918): Mmm.
A (919): No, sen jälkeen taas...
A (921): Mennään tänne, ja tehään se 16 kertaa tää, näin.
H (923): Mmm.
A (924): Elikkä se nyt on käyny, 112.
H (925): Joo, eli sit me ollaan 112 tulostettu viimeks?
A (926): Niin.
H (926): Joo. Ja sitte varmaan samaan tapaan toi loppu?
A (928): Niin, kunnes, kunnes tää luku, kunnes se on saatu jaettua, elikkä mikä se nyt on, ku tää jaetaan kahella, nii... Tänne tulee 8, 4, 2, elikkä tässon se. Elikkä nää lasketaan, nää yhteen, kunnes me ollaan täs kakkosessa.
H (932): Joo, eli jos meillon luvun arvo 2, niin mitä silloin tapahtuu?
A (933): Silloin tää on koko, koko... Koko systeemi loppuu siihen.
H (936): Mmm, mut tulost, käydäänks tää sisempi viel silloin ku se on, on 2 se arvo?
A (939): Ei.
H (939): Joo. Eli tohon viel lisätään 8 ja 4, mut ei sitä 2:ta.
A (940): Mmm.
H (941): Joo, okei. Eli siit tulee yhteensä 112 plus 8 on 120.
A (942): ...120, 124 on se.
H (943): Joo.
A (943): Elikkä tänne tulostuu ykkösestä 124:ään.

Haastateltava B:

B (605): Sisempi for-lause siis loppuu. Ja sitten luku, joka tällä hetkellä on 32, jaetaan taas kahdella, eli siitä tulee 16.
B (608): Ja sitten tarkistetaan taas se ehto, onko se suurempi kuin kaksi, ja tässä tapauksessa se on, mennään taas tekemään sitä sisempää for-lausetta, jossa taas toi muuttuja k, ja onko muuttuja k pienempi kuin luku.

B (612): Se on, ja nyt se printtaa 16 seuraavaa lukua tähän, 96:sta eteenpäin, eli mikä se nyt on 122. Eikun 102. Eikun 112, noni, ynnälaskutkin on vähän heikoilla... 112.
B (616): Ja sen jälkeen se sisempi for-lause loppuu, ja se ensimmäinen for-lause, mennään taas siihen sen ehtokohtaan, ja luku on edelleen, eikun, siis mennään siihen viimeseen kohtaan, jossa luku jaetaan kahdella, ja luvusta tulee kahdeksan.
B (621): Ja sitte mennään taas siihen ehtokohtaan, ja, mm...
B (622): Tartteeks mun käydä tätä kokonaan läpi?
H (623): Jos sä voit kertoo, et mikä on viimenen luku, minkä se sit tulostaa?
B (623): Jaa, sit mun pitää varmaan vähän kirjottaa. Eli tässä kun luku on toistaseks kahdeksan, ni se tulostaa lukuun 120 asti, sitte se tulostaa lukuun 124 asti, ja se on viimenen luku.
H (627): Okei, eli se tulostaa luvut...
B (628): Yks viiva 124.

Haastateltava C:

C (793): Palataan tähän k plus... Taikka lähdetään taas takaisinpäin k plus yhteen, joka on täs tapauksessa sitte niin paljon kuin yks, ja yks on pienempi kuin 32, elikä mennään sama kierto tonne, 32 eteenpäin tosta, mennään yheksänkytä... yheksänkytä...kuuteen astiko me mennään?
H (802): Joo. Sit sen jälkeen?
C (803): Ja sitten mennään tänne, tänne ja tota 96... öö hetkinen mihis mä pääsin?
C (806): Kuut'nkytäneljää elikä tos tapaukses 32 eli silloin tänne tulis se 16.
H (809): Joo.
C (809): Ja lähettäis uudella kiekalla niin kauan ku 16 on suurempi kuin 2, niin taas palataan tähän nollalauseeseen, elikä silloin eletään niin kauan että me saadaan siitä sataankahteentoista asti.
H (815): Joo.
C (815): Silloin tuo on 8 jaettuna kaks on 4, elikä, elikä...
C (816): Ei vaan tää on lopputulos jo. 8 on suurempi kuin 2, mennään jälleen tänne, mennään 120 asti tol laskurilla, silloin, silloin tonne tulee 4.
C (820): Se tehdään viel kerran 124:ään asti, silloin 2 > 2 ei pidä paikkaansa, jolloin tää hyppää näitten, unohtaa nää kaarisulut ja hyppää tonne.
H (826): Joo, eli?
C (826): Jolloin ohjelma loppuu.
C (826): Elikä silloin se tulostaa 124 se kone viimeisenä.

4.1.3. Viikko 3

Viikon 3 aikana luennoilla oli käsitelty taulukoita ja erilaisia ohjelmointiteknisia asioita, jotka liittyvät taulukoihin. Käsiteltiin sitä, että taulukko itse on olio ja että taulukon alkioina voi olla olioita. Lisäksi oli käsitelty binäärihaku ja yksinkertaisia järjestämisalgoritmeja. Viikon haastatteluissa koodinäytteinä oli kaksi eri luokkaa. Ensin näytettiin luokka, jossa ei ole muuta kuin yksi metodi, joka järjestää parametrina saamansa kokonaislukutaulukon suuruusjärjestykseen. Tämän jälkeen näytettiin lisäksi toinen luokka, jossa luodaan konkreettisia lukuarvoja sisältävä taulukko ja kutsutaan aiemmin esitettyä metodia niin, että taulukko välitetään parametrina. Koodinäytteen järjestämisalgoritmi oli ohjel-

moitu käyttäen while-lauseita, kun luentomateriaalissa sama algoritmi oli ohjelmoitu käyttäen for-lauseita, joten kooditasolla näyte oli erilainen kuin luentomateriaalissa. Pystyäkseen selvittämään ensimmäisenä näytetyn luokan toiminnan, opiskelijan olisi pitänyt soveltaa ”symbolista simulointia” (luvussa 1.4.2), koska käytössä ei ollut metodille mitään todellista parametria eikä siten konkreettisia lukuarvoja, joita käsitellä. Kun molemmat luokat olivat käytössä yhtäaikaa, metodille oli käytössä todellinen parametri ja siten metodin toimintaa pystyttiin simuloimaan konkreettisilla lukuarvoilla.

Viikolla 3 haastateltavan A selviytyminen ohjelmakoodin lukemista tai tuottamista vaativista tehtävistä oli heikkoa. Haastattelijan pyytäessä A:ta kirjoittamaan esimerkin lauseesta, joka luo uuden olion, A kirjoitti epäröinnin jälkeen:

```
new.olio = this.olio
```

Lauseesta löytyy oikea avainsana `new`, ja lauseen rakenne muistuttaa sijoituslausetta, mutta lauseessa on useita ilmeisiä puutteita ja epäloogisuuksia. Viikon koodinäytteen (liite A3) lukemisessa ei päästy metodin otsikkoa pitemmälle. Koodinäytettä koskeva keskustelu jäi sekavaksi. A esitti arveluita metodin tarkoituksesta ”kokonaisuudessaan”, mutta ei osannut liittää arveluitaan mihinkään yksittäisiin lauseisiin metodissa.

Kysyttäessä määritelmiä käsitteille, A:n esittämät käsitykset kuvastivat pirstaleista ja sekavaa tietoa. Alla olevassa lainauksessa A:ta pyydettiin määrittelemään kapseloinnin käsite. Lainauksesta ilmenee, että A:lla on käsitys siitä, mikä on kapseloinnin tarkoitus, mutta hän ei täsmällisesti selitä kapselointia konkreettisten käsitteiden, kuten metodien, muuttujien ja näkyvyysmääreiden avulla:

> H (182): — — **Entäs sitte kapselointi?** Se tais olla jo muutama luento sitte.
> A (186): Joo, elikkä, eli se on siis sitä että, et sinne ei pääse käsiks, näihin, toimiin, näihin tietoihin, mitä sillon, et niitä ei pysty ohjelmassa muuttamaan.
> H (190): Mmm. Eli siis kuka pääse ja mihin tietoihin?
> A (191): Siis se et, joihin tietoihin ei tavallaan... Sillon ku se olio on luotu, ni sinne pystytään syöttämään vaan tietyn tyyppistä tietoa.
> H (193): Mmm. Eli, eli oliolla on tietoa?
> A (195): Niin.
> H (195): Olio sisältää tietoa. Joo. Ja, näytteleeks olion metodit tässä kapseloinnissa jonkunlaista roolia?
> A (198): No mun mielest ne metodit on nimenomaan se tapa mitenkä se on toteutettu.
> H (199): Kapselointi?
> A (200): Niin mut siis, mut tää on nytte vaan tänhetkinen käsitys.

> H (201): Joo. Entä sitte, sen olion kentillä, tai muuttujilla, ni onks niillä joku rooli?
> A (204): Siis kyllähän niilläki määritellään sitä tiedon tyyppiä, mitä siellä voi olla, koska se voidaan käsitellä sit siit...

Toisessa kohdassa haastattelua A osasi sanoa, että *aksessoreilla* päästään käsiksi olion tietoihin. Siltikään hän ei osannut vastata kysymykseen, onko aksessori sama asia kuin metodi:

> H (226): -- -- voitko määritellä termin aksessori, mitä se tarkottaa?
> A (232): Siis aksessori on se tapa millä tavallaan päästään käsiks sen olion tietoihin.
> H (235): Mmm. Joo. Ulkopuolelta?
> A (236): Niin. Eli, eli konstruktori luo sen, perusrakenteen.
> H (238): Olion?
> A (238): Niin.
> H (238): Joo. Ja?
> A (239): Aksessori antaa sitte tavan muuttaa näitä perusrakenteessa olevia asioita?
> H (240): Joo. Ja oliolla voi olla useita aksessoreja? Joo. Onks aksessorit metodeja?
> A (246): Itseasiassa mä en tiedä.

Muutoin pirstaleisen ja sekavan ymmärryksen lomassa erityisesti kiinnitti huomiota, että A kuitenkin osasi kuvata olioiden viitesemantiikan merkityksen aivan oikein ja lisäksi osasi määritellä null-avainsanan merkityksen oikein.

Lisäksi aivan haastattelun alussa A spontaanisti halusi selostaa viimeaikaisia ajatuksiaan ohjelmista. Hän sanoi esimerkiksi, että ohjelman toimintaa selvitettäessä pitää miettiä aina vain yksi asia kerrallaan, ja että käskyt aina kohdistuvat johonkin tiettyyn asiaan. Hänen selostuksensa mukaan nämä havainnot ovat selvittäneet hänen aiempaa käsitystään, jossa ohjelmissa oli kyse ”jostakin mystisistä käskyistä”. Tämän viikon jälkeen A:n osallistuminen katsottiin keskeytyneeksi, eikä A:n haastatteluja analysoitu enempää.

Viikolla 3 haastateltavan B suoriutuminen haastattelussa oli monelta osin hyvää. Hän osasi keskustella asioista sekä ohjelmakoodin tasolla että abstraktien periaatteiden tasolla. Pyydettäessä kirjoittamaan esimerkki uuden olion luovasta lauseesta, B ei aluksi ymmärtänyt kysymystä. Haastattelijan kehottaessa ajattelemaan erästä kurssilla käsiteltyä esimerkkiä, B kirjoitti:

```
Viljavarasto varasto = new Viljavarasto();
```

B unohti aluksi sekä sulut että puolipisteen lopusta. Haastattelija joutui kahteen kertaan kysymään, puuttuuko lauseesta jotakin, ennen kuin haastateltava sai täydennettyä lauseen kokonaiseksi. Tämä tilanne osoittaa, että B:llä oli käsitys siitä, millainen lause on oikein, mutta syntaksin hallinta ei ollut hänellä vielä automatisoitunut.

Haastattelijan kysyessä koodinäytteestä (luokka TekeeJotain, liite A3), mitkä lauseet kuuluvat sisemmän while-lauseen toistettaviin lauseisiin, B osasi vastata kysymykseen helposti. Kysyttäessä ulomman while-lauseen toistettavia lauseita, B joutui tarkastelemaan ohjelmaa huolellisesti ja tekemään merkintöjä paperille ennen kuin osasi vastata. Tämäkin seikka osoittaa, että B oli omaksunut ohjelmien ymmärtämiseen tarvittavat tiedot, mutta niiden soveltaminen ohjelmakoodiin ei ollut automatisoitunut, vaan vaati huolellista prosessointia. Simuloinnin automatisoitumista käsitellään tarkemmin luvussa 4.3.4.

Koodinäytettä simuloitaessa B ei onnistunut selvittämään ensimmäisenä näytetyn luokan toimintaa. Hän ei osannut jatkaa simulointia eteenpäin kohdasta, jossa ensimmäisen kerran viitattiin parametrina saadun taulukon sisältöön, koska käytettävissä ei ollut mitään todellista parametria eikä siten mitään konkreettisia lukuarvoja, joiden avulla simulointia olisi voinut jatkaa (mikään ei periaatteessa olisi estänyt B:tä itse muodostamasta jotakin esimerkkitaulukkoa simulointinsa avuksi, mutta niin hän ei tehnyt). Kun B sai käyttöönsä koodinäytteen toisen osan, jossa aiemmin käsiteltyä metodia kutsuttiin todellisella parametrilla, hän osasi simuloida metodin suoritusta ja päätellä metodin funktion. Luvussa 1.4.2 esitellyn Détiennen ja Solowayn (1990) tutkimuksen tuloksissa mainittuja simulointistrategioita olivat muiden muassa ”symbolinen simulointi” ja ”konkreettinen simulointi”. Havaintojen perusteella voitiin todeta, että B ei onnistunut käyttämään symbolisen simuloinnin strategiaa, mutta konkreettisen simuloinnin strategian käytössä hän onnistui.

B osasi määritellä kapseloinnin ja aksessorin käsitteet. Niiden määrittelyn jälkeen B:ltä kysyttiin aksessorien määreistä:

> H (128): Joo. Ku meillon nää määritteet public ja private, ni onks aksessorit enemmän jompaakumpaa? Vai voiks ne olla kumpaa tahansa?
> B (131): Mä en itse asiassa tiedä, jos niinkun, mä oon käsittäny et aksessoreiks sanotaan nimenomaan niitä metodeja, joihin pääsee käsiks ulkopuolelta, eli jotka on niitä public metodeja, mutta luokkaan voi kirjoittaa mun käsittääkseni kyllä ihan private metodejakin, mut sitä, kutsutaanko niitä sitte aksessoreiks, ni sitä mä en tiedä.

Tässä lainauksessa B osoittaa taitoa eritellä omien tietojensa rajoja. Hän myös osoittaa kykyä sujuvasti liikkua abstraktimman funktionaalisemman tason käsitteen (aksessori) ja konkreettisten käsitteiden (metodien määreet) välillä.

B ei ymmärtänyt, mitä tarkoittaa *olion tila*:

> (#207)
> H: Tota, entä mikä on olion tila?
> B: Ei hajuakaan, tää tila on mulle aina ollu ihan käsittämätön juttu.

Aikaisemmin, viikolla 1, B osasi helposti määritellä, mitä tarkoittaa *algoritmin tila*.

B ei käyttänyt olioille viitesemantiikkaa vaan arvosemantiikkaa. Tätä väärinkäsitystä käsitellään enemmän luvussa 4.2.4. B raportoi itse, että on kiinnittänyt huomiota terminologisiin ongelmiin opiskelussaan. Terminologiasta lisää luvussa 4.3.1.

4.1.4. Viikko 6

Viikon 6 haastatteluihin mennessä oli käsitelty kurssien koko asiasisältö. Kurssikokonaisuuden jälkimmäisen kurssin eli haastatteluviikkojen 4–6 aikana oli käsitelty uudelleen Java-kielen monet perusasiat, mutta yksityiskohtaisemmin ja teknisemmin kuin ensimmäisellä kerralla. Käsiteltäviä aiheita oli kielen syntaksin perusasiat, muuttujien eri tyypit, monet erilaiset operaatiot, lausekkeet ja lauseet, metodien toiminta ja rekursion käsite. Olio-ohjelmoinnin asioista käsiteltiin luokkien eri jäseniä, periytymiseen liittyviä mekanismeja, rajapinnat ja abstraktit luokat sekä näkyvyysmääreet. Viikon koodinäytteessä oli luokka, jonka kenttinä oli yksi String-muuttuja ja toinen muuttuja, jonka tyyppi oli olion luokka itse. Kyseisen luokan olioista voitiin siis muodostaa yhteen suuntaan linkitettyjä tietorakenteita. Luokassa oli metodi, jolla pystyi tulostamaan tietystä oliosta lähtevän linkitetyn ketjun kaikkien olioiden String-muuttujien sisällöt. Luokan main-metodin koodissa luotiin luokasta kuusi ilmentymää, joita linkitettiin erilaisiksi ketjuiksi. Lisäksi koodissa oli kutsuja tulostusmetodiin siten, että kutsut tulostivat 1–3 olion ketjuja. Luokan toiminnan ymmärtäminen edellytti olioviittausten käsittelyn hallintaa.

Viikolta 6 analysoitiin simulointiepisodit haastateltavien B ja C haastatteluista. Simuloitu ohjelmakoodi löytyy liitteestä A5.

Viikolla 6 B:n haastattelussa koodinäytteen (liite A5) tarkastelun aluksi haastattelija pyysi yleiskuvaa luokasta. B luki luokan koodin suoraviivaisesti tekstijärjestyksessä läpi ja selitti koodin. Kun haastattelija pyysi selvittämään, mitä ohjelma tekee, B mietti pitkään ja silloinkin aloitti koodin käsittelyn tekstijärjestyksessä eli alussa olevista konstruktoreista. Tämä erosi selvästi C:n etenemistavasta, sillä C loi pikaisen yleissilmäyksen joihinkin luokan osiin ja siirtyi hyvin pian suoritusjärjestyksen mukaiseen tarkastelujärjestykseen.

Haastattelussa B ei oma-aloitteisesti juurikaan onnistunut etenemään suorituksen simuloinnissa, eikä rajaamaan ongelmaansa selväksi kysymykseksi tai kuvaukseksi siitä, mikä estää hänen etenemisensä. Tällainen toiminta ongelmanratkaisutilanteessa vastasi sitä, mitä Perkins *et al.* kuvasivat ”pysähtyjän” toimintatavaksi (käsitelty luvussa 1.5.2). Kyselemällä haastattelija sai ongelman rajattua siihen, että B ei osannut soveltaa while-lauseen ehto-osassa olevaa vertailuoperaattoria, jossa verrattiin olioviitettä null-viitteeseen. Kun haastattelija selitti, miten vertaaminen null-viitteeseen käsitellään, B pystyi ilman ongelmia simuloimaan while-silmukan suorituksen jäljellä olevat iteraatiot ja seuraamaan olioviitteitä. Se, että B itsenäisesti seurasi olioviitteitä osoittaa, että ainakin olioviitteillä operoinnin B osasi.

Simuloinnin lopuksi haastattelija pyysi B:tä omin sanoin kuvaamaan, mitä tapahtui metodin suorituksen aikana. B osasi esittää selkeän kuvauksen, joka oli itse tuotettu, siis ei lainausta haastattelun aiemmasta keskustelusta, ja oikein.

C:n simulointi alkoi seuraavasti: C loi nopean yleissilmäyksen luokan koodiin. C alkoi lukea pääohjelmaa ja päätyi tarkastelemaan konstruktoreita, koska ensimmäisenä pääohjelmassa olivat luontilauseet, jotka johtivat konstruktorien suoritukseen. Tämä tapa erosi selvästi B:n tavasta aloittaa koodinäytteen tarkastelu.

Metodin tulostaLista ensimmäisen suorituskerran simulointina C päätyi tulokseen ”Olettaisin, et se tulostaa haa pilkku hap.” Tähän päätyäkseen C ei kiinnittänyt huomiota tulostaLista-metodissa olevan while-lauseen iterointiin, mutta while-lauseen sisällä olevat lauseet hän käsitteli. C ei siis käsitellyt huolellisesti ohjelman lauseita suoritusjärjestyk-

sessä, vaan turvautui jonkinlaiseen heuristiseen simulointiin, jossa hän pääpiirteissään seurasi ohjelman suoritusjärjestystä, mutta jätti joidenkin lauseiden merkityksen huomiotta.

C:n esittämä yllämainittu tulos ei ollut oikein, ja haastattelija mainitsi tästä. C yritti simuloida metodin toimintaa ja luki ääneen while-lauseen: ”kun nykyinen on erisuuri kuin nolla”, mutta ei simuloinut silmukan iterointia. C:n käyttämä sana ”nolla” herättää epäilyksiä, mutta toisaalla haastattelussa C lukee ”String sisältö on sisältö, seuraava on null. Okei, seuraavaa siinä ei ookaan.”, mikä osoittaa hänen ymmärtävän null-avainsanan merkityksen oikein.

Seuraavaksi C pohti omaa hypoteesiaan, että jokin operaatio muuttaisi jonkin olion `sisältö`-muuttujan sisältöä niin, että sinne tulisi yhden sijasta kaksi sanaa. Hän päätyi tulokseen: ”Nii jolloin se muuttais tuota haa-sanaa, että haa plus hap, niin siihen tulis haa-hap hip. Ei voi tulla noin. Itseasias voi. Mä muuttaisin tänne, tän ekan nyt niin että se tulostaa myös tuon hip-sanan sinne.” Hänen esittämänsä muutos tulostukseen vastasi ohjelman oikeaa toimintaa, mutta hänen ajatuksensa siitä, miten tähän tulokseen päädytään, oli väärä. C ei pyydettäessä osannut selittää, minkä operaatioiden kautta `sisältö`-merkkijonojen katenointiin päädyttäisiin, ja hän itsekin epäili omaa käsitystään. C:n toiminta ongelmanratkaisutilanteessa vastasi sitä, mitä Perkins *et al.* kuvasivat luvussa 1.5.2 käsitellyssä tutkimuksessa ”liikkujan” toimintatavaksi.

Vasta aivan haastattelun lopussa, kun haastattelija oli maininnut, että `tulostaLista`-metodissa oleva silmukka käydään useaan kertaan läpi, C esitti päätelmän: ”Niin sen tähden täällä on periaatteessa toi null-arvo, koska se käy niin kauan tuon, tuon että tulee tää null vastaan myös.” Tämä osoittaa hänen ymmärtävän while-silmukan iteroinnin merkityksen, jonka ymmärtäminen aiemmin olisi ollut avain ohjelman oikean toiminnan ymmärtämiseen.

C ilmaisi kvalitatiivisen päätelmän siitä, että kun ohjelma alkaa tulostaa jotain, ohjelma on ensin suorittanut luontilauseet: ”...jos tää ohjelma alkaa tulostelemaan, ku tää ohjelma ajetaan, ni se on luonu ensin nää uudet, uudet Lista-oliot, sit se printtaa että haa...”

4.2. Ohjelmointikielen käsitteisiin liittyvät havainnot

4.2.1. Käsitys tyypistä rajoitejoukkona

Haastateltava A esitti sekä viikolla 1 että viikolla 3 käsityksiä muuttujista ja muuttujan tyypistä. Viikolla 1 häneltä kysyttiin muuttujan määritelmää:

> H (247): Okei. No, entä sitte muuttuja?
> A (249): Muuttuja on se tyhjä laatikko, eli se voi saada mitä tahansa arvoja. Tai siis **jos se muuttuja on määritelty**, niin sit se voi saada tietyn tyyppisiä arvoja.
> H (252): Joo.
> A (253): **Jotka on päätetty etukäteen mitä ne voi olla.**
> H (253): Joo. Millanen asia se muuttuja on tietokoneen kannalta?
> A (256): Emmä pysty ymmärtämään millanen nyt sit... Siis mä, no se muuttuja on aina, sillä on aina yks arvo, että mikä se sitte on, ja sitte ku se arvo muuttuu, ni sitten se on se.

Lainauksessa A esittää muuttujan määrittelemisen ja siten muuttujan tyypin ehdollisena. Lainauksen toinen korostettu kohta mahdollisesti viittaa samaan ajatukseen, jonka A esittää viikolla 3 (lainaus alla), jossa hän lukee mahdollisten arvojen arvojoukon rajaamisen mukaan tyypin määritelmään. Viikolla 3 kapseloinnista puhuttaessa haastattelija kysyi A:lta yksitellen ensin metodien roolista kapseloinnissa ja sitten olion muuttujien merkityksestä kapseloinnissa:

> H (195): -- -- näytteleeks olion metodit tässä kapseloinnissa jonkunlaista roolia?
> A (198): No mun mielest ne metodit on nimenomaan se tapa mitenkä se on toteutettu.
> H (199): Kapselointi?
> A (200): Niin mut siis, mut tää on nytte vaan tänhetkinen käsitys.
> H (201): Joo. Entä sitte, sen olion kentillä, tai muuttujilla, ni onks niillä joku rooli?
> A (204): Siis **kyllähän niilläki määritellään sitä tiedon tyyppiä**, mitä siellä voi olla, koska se voidaan käsitellä sit siit..., tai siis voidaan määritellä se et minkä tyyppist, tyyppisiä muuttujia siellon, ni sillon voidaan syöttää, ja sitten näille muuttujille voidaan antaa, antaa...
> H (209): Mmm.
> A (209): Antaa... Jos mä nyt muistan oikein. Sanotaanks se niinku metodeilla vai lauseilla se, kuinka rajataan niitä arvoja mitä se muuttuja voi saada?
> H (212): Nyt mä en oo ihan varma mitä sä kysyt.
> A (213): No mä yritän selittää. Esimerkiks jos, jos, siis tää nyt varmaan on ollukki jo pari luentoa sitte, mutta kun mä en muista että, jos jossain, **jos halutaan että se saa vaikka arvot yks viiva kakstoista**.

A:n mukaan siis sekä metodeilla että muuttujilla määritellään tiedon tyyppiä. Hänen käsityksensä mukaan siis useampi kuin yksi asia voi määritellä tiedon tyyppiä. Tämä vaikuttaa

kuvastavan käsitystä tyypistä jonkinlaisena rajoitejoukkona, jonka avulla rajataan sitä, mitä tai millaista tietoa muuttuja voi sisältää.

Koodinäytteen tarkastelun yhteydessä haastateltava joutui tarkastelemaan koodia, joka oli hänelle jopa syntaksin tasolla vierasta. Vaikuttaa siltä, että hän yrittää aluksi soveltaa rajoitemalliaan itselleen vieraaseen syntaktiseen merkintään, ennen kuin muistaakin merkinnän tarkoituksen:

> A (552): Eli siis tässon nyt, jos ajatellaan niinku että, jos mä ajattelisin, koska mä en nyt oo, en oo perehtyny tähän, mutta mä ajattelisin että tää, **nää hakasulkeet viittaa siihen että minkä tyyppisiä arvoja tässä voidaan käyttää**.
> H (556): Mmm.
> A (557): Ei vaan siihen, et on kyse taulukosta.

A vaikuttaa käyttävän tyypin käsitteestä jonkinlaista ”rajoitejoukkomallia”, jossa tyyppi tarkoittaisi jonkinlaista, mahdollisesti dynaamista, tapaa määrittää rajoitteita muuttujan arvolle. Todellisuudessa rajoitejoukkomalli muuttujista on väärä. Muuttujan tyyppi on muuttujalla oleva yksittäinen ominaisuus, jonka tarkoitus on kertoa, millaista tulkintaa muuttujan sisältämään informaatioon voi soveltaa.

Käsiteltäessä viikon 1 koodinäytettä (liite A1), A myös virheellisesti katenoi tulostuslauseessa toisiinsa muuttujien nimiä, vaikka olisi pitänyt katenoida muuttujien arvoja. Tämän jälkeen A käsitteli sijoituslauseen ”`iso = pieni + iso;`” katenointina, vaikka molemmat muuttujat ovat int-tyyppiä. Haastattelija yritti kiinnittää huomiota muuttujien esittelyssä esiintyviin tyyppeihin, mutta A esitti seuraavan perustelun:

> (#409)
> H: Mitenkäs nää on esitelty nää muuttujat?
> A: Niin mutta siis, siis tossa äskeisessä laskuharjoituksessa kävi selville, että int, integer, elikkä siis kokonaisluku, voi muuttua liukuluvuksi. Ni tässä, jos tää menis analogisesti, ni silloin tää olis muuttunut merkkijonoks, tää.
> H: Eli se muuttujan tyyppi olis muuttunut tossa?
> A: Niin, niin siis mä ajattelin et siinä tapauksessa se olis sallittua, jos se voi muuttua liukuluvuksi, mutta mä en tiedä.

Haastateltava siis esitti **käsityksen, että muuttujan tyyppi voi muuttua**. (Joissakin kielissä tämä polymorfismin laji on olemassa, mutta ei Javassa.) Virhekäsitys saanee alkunsa tyyppimuunnoksen käsittelystä, jossa sijoitettaessa int-muuttujaa liukulukumuuttujaan on

tehty tyyppimuunnos. A on siis sekoittanut yksittäisen arvon muuntamisen siihen, että muuttujan tyyppi vaihtuisi.

A:n virheellinen tyypin käsite sisälsi siis seuraavat ominaisuudet: 1. Muuttujan tyyppi on ehdollinen, voisi siis olla muuttujia, joille ei ole määritelty tyyppiä. 2. Muuttujan tyyppi saattaa muuttua ohjelman suorituksen aikana. 3. Tyypin käsite sisältää tietotyypin (”arvolajin”) lisäksi määrityksen arvojoukosta. 4. Sekä muuttujat että metodit osallistuvat ”tiedon tyypin” määrittelemiseen.

4.2.2. Käsite pääohjelma käsitteen luokka korvaajana

Viikon 3 haastattelussa A ei omasta aloitteestaan käyttänyt tai maininnut käsitettä *luokka* lainkaan. Sen sijaan hän käytti käsitettä *pääohjelma* joissakin sellaisissa kohdin, joissa *luokka*-käsitteen käyttö olisi ollut sopivaa.

Seuraavassa katkelmassa viikon 3 haastattelusta A määritteli näkyvyysmääreen merkityksen suhteessa pääohjelmaan:

> A (289): -- -- Koska private oli se, mihinkä ei voinu päästä käsiks mistään, ulkoapäin, ja publiciin taas pysty ja siinoli sit tää, arvoa ei voi muuttaa jossei siihen päästä käsiks ulkoa.
> H (295): Joo. Mitä tää päästä käsiks tarkottaa?
> A (296): Siis... Siis se tarkot... Jos mä nyt muistan oikein, niin, niin, pääohjelmaa.
> H (299): Mmm.
> A (300): Pääohjelmalle voidaan antaa erilaisii metodinkutsuja. Liittyyks tää nyt tähän?
> H (303): Joo.
> A (303): Niin **siis nää public on sellasii mitä se pääohjelma pystyy käyttään, mut toi private ei**.

Toisessa saman haastattelun kohdassa käsiteltiin sanallista tehtävää (Tehtävä 2, liitteessä B2). Haastateltava ymmärsi, että main-metodi on pääohjelma, mutta luokan nimeä hän kuvasi ”pääohjelman nimeksi”:

> A (678): Elikkä ensin kirjoitetaan se pääohjelma.
> H (679): Joo. Ja miten se...?
> A (679): public static. Ei vaan siis siis ensin tulee **pääohjelman nimi**.
> *H lukee ääneen, mitä A on kirjoittanut paperille:*
> H (680): Mmm. Eli public class.
> A (681): Public class, joku, joku, lukuarvo, vaikka.
> H (683): Joo. Ja sitte?

A (683): Ja sit tulee pääohjelma.

Lainauksessa haastateltava A selvästi käyttää ”pääohjelman” käsitettä olennaisena ohjelmien rakenteen osana, jolla olisi myös sellaista merkitystä, että se määrää esimerkiksi näkyvyysmääreiden merkitystä. Tämä heijastaa sitä, että kurssin oppimateriaalissa ja tehtävissä käsite ”pääohjelma” on ollut paljon esillä. Tämä on esimerkki käsityksestä, joka rikkoo luvussa 1.1.3 esiteltyä no-function-in-structure-periaatetta vastaan, koska ”pääohjelma” on funktionaalisen tason käsite, eikä lainkaan sellainen ohjelman rakenteellinen osa, että näkyvyysmääreiden merkitystä voitaisiin erikseen määritellä suhteessa siihen.

4.2.3. Olion ja luokan käsitteet

Viikon 6 haastattelussa C:n ymmärrys olioiden kentistä ei ollut täsmällistä: kysymykseen ”...montaks kenttää jokaisella Lista-oliolla on?” hän vastasi ”Periaatteessa jokaisella Lista-oliolla on vähintään kaks kenttää.” Myöhemmin C täsmensi, että kenttiä on tasan kaksi ja osasi nimetä kentät, mutta sana ”vähintään” ilmaisee, että hänen käsityksensä kenttien lukumäärän määräytymisestä ei ollut täsmällinen. Tieto kenttien lukumäärästä on helposti ja yksikäsitteisesti luettavissa kyseisen luokan ohjelmakoodista.

C:n vastaukset olion kentistä herättivät kysymyksiä siitä, millainen mentaalinen malli C:llä oli luokista, olioista ja olion tilasta. Näitä asioita selvitettiin viikon 4 haastattelusta, jossa C:tä pyydettiin määrittelemään vuorollaan käsitteet luokka, olio ja olion tila. **Seuraavassa C:n käsitteenmäärittelyitä viikolta 4:**

Luokan käsitteestä C lausui seuraavasti: ”Se on periaatteessa niitten tämmönen hierarkian ylin osa ja se saattaa sisältää olioita, metodeita, konstruktoreita, mitä tahansa, niin et ne kuuluu kaikki johonkin luokkaan aina.” Käsitys siitä, että luokka olisi hierarkian ylin ”osa” ja että oliot ja metodit kuuluisivat luokka-käsitteen alle jotenkin rinnasteisina, ilmentää selvästi epätäydellistä käsitystä olion käsitteestä. Tämä on yhdenmukaista olion käsitteen ja olion tilan käsitteiden määrittelyn kanssa.

Kysyttäessä *olion* määritelmää C vastasi: ”Mä en tiedä lukiko jossain luokan ilmentymä, mut mä en tiedä mitä se tarkoittais mun mielessä.” Hän siis osasi toistaa oppimateriaalin

määritelmän oliosta, mutta ei ymmärtänyt sen merkitystä. Määritellessään oliota omin sanoin C määritteli, että olio on ”toimintamalli”, joka ”muodostuu mahdollisesti jostain toiminnasta, lausekkeista, palautelausekkeista ja niin pois päin”. Olion määritelmässä C:llä ei ollut mukana käsitystä, että oliolla olisi jokin tietosisältö.

Olion tilan määritelmää kysyttäessä C määrittelee olion tilan asiana, joka ohjelmaa suoritettaessa aina jokaisen lauseen jälkeen tarkistetaan. C mainitsee useaan kertaan toiminnan ”vaiheen”: ”missä vaiheessa kokonaisuutta mennään”, ”missä vaiheessa milloinkin ollaan” ja ”missä vaiheessa olio on”. Olion tilaa määritellessään C ei viitannut mitenkään olion tietosisältöön, vaan pelkästään toiminnallisiin yksiköihin, kuten lausekkeisiin ja palautusarvoihin.

C:n käsitykset kurssikokonaisuuden neljänneltä viikolta olivat selvästi puutteellisia. Niihin ei sisältynyt selkeää käsitystä luokkien ja olioiden suhteesta, eikä käsitystä siitä, että kullakin oliolla on oma tietosisältö, joka määrittelee olion tilan. C:n käsitys, että luokka sisältää ”olioita, metodeita, konstruktoreita, mitä tahansa” on selvästi väärä. Luokka ei sisällä olioita, vaan oliot ovat luokan ilmentymiä. Tämän C osasi toistaa ulkoa opittuna määritelmänä, mutta itsekin totesi, ettei tiedä, mitä määritelmä tarkoittaa.

Viikolla 3 haastateltava B ei osannut sanoa yleistä määritelmää oliolle, mutta osasi kapseloinnin määrittelyn yhteydessä selittää, että kapseloitu informaatio on tallennettuna olion sisäisiin kenttiin eli private-kenttiin, ja että tähän kapseloituun informaatioon päästään käsiksi julkisten metodien eli aksessorien avulla. Viikolla 3 B ei kuitenkaan osannut vastata kysymykseen, mitä tarkoittaa *olion tila*. Kapseloinnin määritelmän perusteella B:n käsitys olion käsitteestä oli selkeä ja konkreettinen ja olisi kuviteltavissa, että ymmärrys olisi silloin helposti laajennettavissa koskemaan *olion tilan* käsitettä. Olion tilaa oli käsitelty kurssilla jo aikaisin olioiden yhteydessä ja käsitettä oli myös verrattu *algoritmin tilan* käsitteeseen, joka oli käsitelty aivan kurssin alussa ennen olioiden käsittelyä. Viikolla 1 B osasi määritellä, mikä on algoritmin tila.

Olioihin ja luokkiin liittyviä käsityksiä on raportoitu kirjallisuudessa. Eckerdalin ja Thunén (2005) tutkimusta ja Hollandin *et al.* (1997) tutkimusta käsiteltiin luvussa 1.5.4. Luvussa 5.2.2 pohditaan tarkemmin nyt saatujen tulosten ja muiden tutkimusten suhdetta.

4.2.4. Olioiden viitesemantiikka

Viikon 3 haastattelussa B selviytyi simulointitehtävästä ja hänen vastauksensa kuvastivat monessa tilanteessa johdonmukaista ja täsmällistä ymmärrystä. Tästä huolimatta B käytti ajattelussaan johdonmukaisesti olioille arvosemantiikkaa eikä viitesemantiikkaa, joka olisi ollut oikein. Toinen haastateltava, A, ei päässyt simulointitehtävässä edes alkuun, ja muutoinkin hänen käsityksensä kuvastivat epäselvyyttä monista keskeisistä käsitteistä, mutta A osasi selittää olioviitteiden käytön selkeästi. Tässä luvussa tarkastellaan yksityiskohtaisesti B:n ymmärrystä olioviitteistä ja lopuksi A:n selitystä olioviitteistä.

Haastateltava B ymmärsi, että Stringit ovat olioita ja että uutta Stringiä luotaessa luodaan uusi olio. Lisäksi B osasi erottaa toisistaan muuttujan esittelyn ja arvon sijoittamisen muuttujaan:

> H (309): Joo. Entä oisko sul mieles muita asioita, tai, tai muita esimerkkejä lauseesta, joka luo uuden olion? Jos mietitään et ihan aika alussa oli jo puhe näistä stringeistä, että string on aina olio, niin tota, voisko sen perusteella keksiä toisenlaisen esimerkin lauseesta, joka luo uuden olion?
> B (317): Hmm. No varmaan jos, jos mä nyt teen, tai siis esittelen string-tyyppisen muuttujan, niin se luo sitten string-olion.
> H (326): Luoks se sen sillon kun se muuttuja esitellään, eli jos sä sanot ihan vaan että ”String jono;”, siinähän se niinku esitellään se muuttuja...
> B (330): Mun käsittääkseni, nyt menee kyllä arvauksen puolelle, mut **mun käsittääkseni se luo sen olion silloin ku sille muuttujalle annetaan joku arvo**.
> H (332): Joo.
> B (332): Tai se saa jonkun arvon.
> H (333): Joo. Eli jos me sanotaan vaikka ensin että String jono1 on ja sit jotain, lainausmerkeis tekstiä...
> B (336): Niin sillon se luo siinä vaiheessa, sen on-merkin jälkeen, ehkä, vähän vaikea sanoa, mut siinä vaiheessa sen olion.

Haastateltavan mentaalisen mallin mukaan siis oli selvästi mahdollista, että on olemassa esitelty muuttuja, joka mahdollisesti voi ottaa arvokseen olion, mutta jolla ei vielä ole oliota arvonaan. Tällaisessa tilanteessa muuttujan arvona on null-viite, mutta haastateltava ei tuntenut kyseistä avainsanaa. Haastattelija kysyi null-avainsanasta sen jälkeen, kun haastateltava oli määritellyt, mitä tarkoittaa oliotyyppinen muuttuja (haastattelussa aiemmin kuin edellinen lainaus):

> H (192): Mmm. Joo. Liittyyks tähän sit tommonen sana kun null?
> B (194): Juu, kyllä varmasti, mutta en osaa selittää...

> H (195): Suunnilleen et mitä se tarkoittaa?
> B (196): Ei pienintäkään hajua.

Jos haastateltava käyttäisi viitesemantiikkaa ajattelussaan, tämä saattaisi tulla esille siinä, miten haastateltava määrittelee *oliotyyppisen muuttujan* tai *alkeistyypin* käsitteitä. Seuraavissa kahdessa lainauksessa näistä asioista kysytään, mutta B:n käsitykset eivät ilmennä viitteiden käyttöä:

> H (182): Entä sit mikä, mitä tarkoittaa oliotyyppinen muuttuja?
> B (187): No se on siis muuttuja, jonka tyyppi on, tai siis se on muuttuja, joka voi saada arvokseen ainoastaan sen olion, jonka tyyppi se on, tai sen tyyppisen olion.

> H (196): Okei, tota, entä, mitä tarkoittaa alkeistyyppi?
> B (198): Enpä tiedä sitäkään.
> H (199): Mmm. Oisko esimerkiks, öö, jos aatellaan et meillon vaikka int ja sit meillon joku olio, niin kumpi niistä saattais olla alkeistyyppi?
> B (202): No se int on alkeistyyppi, mut en mä osaa, siis, sitä, sen tarkemmin määrittää, tai siis, en pysty selittämään, mitä se tarkoittaa. Tiedän, että jotkut tyypit on alkeistyyppejä ja jotkut tyypit ei, mutta se mikä se ero on, niin enpä tiiä.

Yllä olevassa lainauksessa B osasi määritellä, mikä on *oliotyyppinen muuttuja*, mutta esitti määritelmän vain tyypin kannalta eli niin, että tyyppi rajoittaa sitä, minkä olion muuttuja voi saada arvokseen. B tiesi int-tyypin olevan alkeistyyppi, mutta ei osannut selittää, miten alkeistyypit eroavat muista tyypeistä. Kummankaan käsitteen määrittelyn yhteydessä B ei maininnut olioviitteistä mitään.

Hieman myöhemmin haastattelussa haastattelija selvitti esimerkin avulla, käyttikö B olioista arvosemantiikkaa vai viitesemantiikkaa. Tämän luvun ensimmäisessä lainauksessa nähtiin, että haastateltava ymmärsi Stringien olevan olioita. Haastattelija kirjoitti paperille:

```
String jono1 = "tekstiä";
String jono2 = jono1;
```

Tämän jälkeen haastattelija kysyi:

> H (354): Eli meillon String jono1 saa arvokseen tekstiä lainausmerkeissä. Joo. Ja sit meillon String jono2 saa arvokseen jono1. Niin, ni luoks tää jälkimmäinen lause uuden olion?
> B (364): Vaikee sanoa. Kyllä se mun käsittääkseni luo.
> H (365): Joo. Eli sillon se, se lois uuden olion, jonka sisältö on sama ku toi jono1.
> B (368): Joo.

B ei ollut varma omasta ymmärryksestään, mutta vastasi kuitenkin niin, että hänen vastauksensa ilmentää arvosemantiikkaa olioille, mikä on väärä vastaus. Toinen mahdollisesti olioiden viitesemantiikkaan liittyvä kysymys tuli esille aivan haastattelun lopussa, haastattelijan kysyessä teeJotain-metodin (liite A3) palautusarvosta:

> (#893)
> B: Ja se palauttaa sen taulukon, järjestetyn taulukon arvonaan.
> H: Mmm. Onks se se sama taulukko, vai onks tää luonu uuden taulukon?
> B: Se on se sama taulukko.

Voidaan ajatella, että ilman viitesemantiikkaa saman taulukon palauttaminen ei olisi mahdollista. Jos taulukko-olio olisi välitetty parametriksi arvosemantiikkaa käyttäen, taulukosta olisi jouduttu luomaan kopio. On kuitenkin mahdollista, että kysymykseen vastatessaan haastateltava pohti, luotiinko metodissa eksplisiittisesti uutta taulukkoa vai ei. Metodissa ei luotu uutta taulukkoa. Näin B saattoi päätyä tulokseen, että taulukon on oltava sama. Tätä tulkintaa tukee tapa, jolla haastateltava määritteli oliotyyppisen muuttujan ilman mitään mainintaa viitteiden käytöstä. Edellä B ajatteli, että sijoitettaessa oliotyyppistä arvoa muuttujaan luotiin uusi olio, mutta teeJotain-metodissa ei tehdä tällaista sijoitusta, vaan pelkästään palautetaan parametrina saatu arvo return-lauseella.

Toinen haastateltava, A, lausui, että muuttujan arvona voi olla viite olioon:

> H (350): Onks oliotyyppinen muuttuja sama asia ku viite olioon?
> A (353): Ei se mun mielest oo sama asia.
> H (354): Mmm. Mitä eroa niillä sitte olis?
> A (356): Siis jonkun **muuttujan arvo on viite olioon**.

Lisäksi A pystyi määrittelemään null-avainsanan merkityksen erittäin selkeästi:

> H (513): -- -- onks luennoilla tullu esiin sellainen ku null, asia ollenkaan?
> A (517): Aa, toi on jotenkin hämärästi tuttu, mutta jos se oli viime luennolla, ni... Elikkä oliks se niin, eiku, eiku se oli, oliks nyt tätä näin, et ku joku saa arvon ei mitään, mutta en mä muista mikä oli ei mitään ja mikä oli niinkun...
> H (520): Mikä saa arvon?
> A (521): Siis, siis viite, siis olio, **olion viite, ei johda mihinkään, ni se oli muistaakseni null**.

Näissä lainauksissa A osoitti ymmärtävänsä olioiden viitesemantiikan ainakin funktionaalisten määritelmien tasolla.

4.2.5. Terminologiset ongelmat

Viikon 3 haastattelussa B oma-aloitteisesti kertoi kokemuksiaan siitä, että terminologia aiheutti hänen opiskelulleen ongelmia. Haastattelun alussa haastattelija kysyi harjoitustehtävien tekemisen sujumisesta ja B mainitsi, että kurssilla käytettävä terminologia aiheutti ongelmia, vaikka kyse ei ollut pelkästään ohjelmointitermeistä:

> H (27): Onks joku erityinen juttu jääny epäselväks?
> B (28): No ei oikeestaan. Enemmän ne on ehkä semmosia, jotain tehtävien sanamuotoja, joita ei välttämättä oo oikein ymmärtäny, jossain tilanteessa.
> H (30): Oisko sul esimerkkiä, et millasia sanamuotoja?
> B (30): No siis, viime laskareissa oli tehtävä jossa piti muuttaa string-jono numeroita kokonaisluvuks, ja siinä pähkäilin pitkään, kun siinä tehtävän selostuksessa oli hirveesti, toisiaan lähellä olevia käsitteitä, ja siinoli hirveesti sit semmosia sanoja, jotka viittas eri käsitteisiin, mut lähes samoilla sanoilla, et oli numeromerkki ja numeromerkkiarvo ja numero ja luku. Tämmösiä siinä tehtävänannossa peräkkäin, niin meni aika pitkään aina välillä siihen että mitäs, niinku mihinkä tässä nyt viitataan ku sanotaan luku, ja mihinkä tässä nyt viitataan ku sanotaan numeromerkki.

Lisäksi haastateltava sanoi, että ohjelmointiterminologia yleensä aiheutti ongelmia:

> B (43): Mmm, no siis ihan yleisesti nää, niinkun terminologia mitä käytetään näistä ohjelmoinnin, tai siis, tän kurssin puitteissa, mitä on oppinu näistä ohjelmointitermeistä, mitä nyt on, metodit ja aksessorit ja kentät ja tyypit ja muuttujat, tämmöset menee niinkun, kun niitä tulee niin paljon, ja suurin osa on täysin uutta terminologiaa, tai sitten tuttu sana, joka viittaa täysin uuteen asiaan, tai täysin toiseen asiaan ku mihin sen on tottunu viittaavan.
> B (51): Niin sit ne vie aikaa kun tehtävänannossa, sit niinku laskaritehtävässä, lukee, ja kun käsketään käsittelemään jotain asiaa tietyllä tavalla, ni sen kelaamiseen, että mitäs tää nyt tarkottikaan.
> B (54): Ja sitte vaikka puhutaan kentistä tai vastaavista, niin sitten, se että yleensä täytyy aina konsultoida sitä materiaalia, että mitä se tämä nyt tarkottikaan tämä kenttä tässä yhteydessä, esimerkiks.

Jatkossa haastattelija kysyi, onko luentojen seuraaminen onnistunut hyvin. Haastateltava kertoi, että konkreettisten vertauskuvien käyttö häiritsi abstraktien käsitteiden oppimista:

> B (75): Joo. Tai siis, siinon se, että ainaki mulla se fiilis on semmonen et ne, mä en hirveesti pidä siitä liiallisesta konkretisoimisesta, koska se jotenkin, siinä helposti mulla ainakin käy niin, että ne mielikuvat jää sille konkretian tasolle, kun se on hirveen konkreettista se, ne esimerkit ja ne vertaukset, niin sit, **siit on vaikee päästä irti jotenki siitä konkretiasta tajuumaan se abstraktin tason siitä**.
> B (81): Et mulle puhutaan koko ajan oliosta koneena, esimerkiks, niin mulle muodostuu niinku mielikuva koneesta, eikä välttämättä mielikuvaa siitä abstraktista asiasta, mikä on olio, ja kaikki ne muut mitä siihen kuuluu.
> H (84): Joo.
> B (84): Et se kyl sotkee, ainakin mua.

> H (85): Joo. Oisko sul vielä mielessä jotain esimerkkejä termeistä tai puhetavoista, jotka häiritsee...?
> B (87): No, tämä, kone-vertaus, esimerkiksi, mistä mä nyt äskenki puhuin, että luokka on koneen piirustukset ja olio on sitten se kone itse, ja metodit on koneen kahvoja ja namiskuukkeleita, ja näin edelleen ja näin edelleen, ni se on yks semmonen mikä mua sotkee, tai sotki pitkään, koska jotenkin sen huomas, että kun aluks yritti niinkun käsittää sen näillä vertauskuvilla, niin sit se jäi, sit sitä ei käsittäny ollenkaan, koska mä koko ajan tiesin, että tässä ei oo kyse mistään koneesta, mä en oikein päässy siihen sisään et mitä tässä haetaan tässä, niinku sen konkretisoimisessa koneeks.

Hieman myöhemmin haastattelussa saatiin konkreettinen esimerkki siitä, miten jotkin termit tuottivat vaikeuksia. Haastateltava ei muistanut, mitä tarkoitti haastattelijan kysymyksessä ollut sana *kenttä*. Tuolloin haastattelija selitti sanan merkityksen (”olion muuttuja”), mutta hieman myöhemmin haastattelussa B käytti kenttä-sanaa merkityksessä ”olio”, mikä oli väärin. Näistä seikoista huolimatta B oli selvästi onnistunut muodostamaan selkeän ymmärryksen myös monista abstrakteista käsitteistä. Tarkasteltaessa, missä yhteyksissä viikon 3 haastattelussa B käytti termejä *luokka* ja *olio* havaittiin, että B käytti niitä aina oikeassa asiayhteydessä.

4.3. Mentaalisten mallien luomiseen ja käyttöön liittyvät havainnot

4.3.1. Yliyleistäminen yhden esimerkin perusteella

Viikolla 3 haastateltava A esitti käsityksen, että olio itsessään voi olla muuttuja. Käsitys on peräisin samasta haastattelusta kuin luvussa 4.2.1 esitetty A:n käsitys siitä, että tyypin käsite sisältää myös määrityksen muuttujan arvojoukosta. Tässä luvussa esitetään, miten A:n käsitys, että olio itsessään voi olla muuttuja, oli syntynyt yhdestä esimerkistä tehdyn yliyleistämisen kautta:

> H (337): -- -- Kerrotko mikä on oliotyyppinen muuttuja?
> A (342): Hmm. Siis mun käsityksen mukaan, jos mä oon ymmärtäny oikein, niin **olio on tavallaan muuttuja**.
> H (345): Olio?
> A (346): Niin.
> H (346): Joo. Eli...?

A (348): Eli, mutta sit mä en pysty niinku ajattelee sitte enempää, koska se vaatii jonkun-näköistä näkökulman muutosta.
H (350): Joo. Onks oliotyyppinen muuttuja sama asia ku viite olioon?
A (353): Ei se mun mielest oo sama asia.
H (354): Mmm. Mitä eroa niillä sitte olis?
A (356): Siis jonkun muuttujan arvo on viite olioon.
H (359): Mmm.
A (360): Mut sen sijaan sitten sillä on sen, se arvo...
H (362): On viite olioon?
A (362): Niin, mutta silloin jos se **olio on itsessään muuttuja**, ni silloin se saa, sen arvoks tulee jotain muuta. Siis muuttuja saa olla...
H (365): Joo.
A (366): Ni sillon, mun käsityksen mukaan ni se ei voi olla sama asia.
H (368): Eli silloin jos olio itse on muuttuja...?
A (369): Niin ni se, ni silloin **se saa niinkun arvoja**.

Tätä käsitystä analysoitaessa syntyi vaikutelma, että monet A:n käsitykset oli johdettu eräästä kurssin oppimateriaalissa olevasta esimerkistä (Wikla, 2001, s. 52):

```
public class KuuLaskuri {

   private int kuu;   // sallitut arvot 1,..12

   // ----- konstruktori: -------

   public KuuLaskuri() {
     kuu = 1;
   }

  // ----- aksessorit: -----------

  public int moneskoKuu() {
    return kuu;
  }

  public void seuraavaKuu() {
    ++kuu;
    if (kuu == 13)
      kuu = 1;
  }
}
```

Jos opiskelijan ymmärryksessä mallina oliosta on olio, jolla on yksi kenttä, niin kentän saamat arvot on helppo samastaa siihen, että olio saa arvoja. Viikon 3 haastattelussa A esitti käsityksen, että olio itsessään saa arvoja. Jos opiskelijan käsitys muuttujasta on, että muuttuja saa arvoja, niin sekaannus, että olio on itsessään muuttuja, lienee varsin ymmärrettävä. Tällainen käsitys on esitetty myös kirjallisuudessa (Holland *et al.*, 1997).

Konkreettinen todistus siitä, että A todella käyttää mainittua esimerkkiä ajattelussaan, saatiin viikon 3 haastattelussa (sama kohta on lainattu myös luvussa 4.2.1):

> A (209): Antaa... Jos mä nyt muistan oikein. Sanotaanks se niinku metodeilla vai lauseilla se, kuinka rajataan niitä arvoja mitä se muuttuja voi saada?
> H (212): Nyt mä en oo ihan varma mitä sä kysyt.
> A (213): No mä yritän selittää. Esimerkiks jos, jos, siis tää nyt varmaan on ollukki jo pari luentoa sitte, mutta kun mä en muista että, jos jossain, **jos halutaan että se saa vaikka arvot yks viiva kakstoista**.

KuuLaskuri-olion, jonka koodi on lainattu tässä luvussa, ainoa muuttuja voi saada arvot 1–12.

4.3.2. Työskentelyjärjestys skeemaa luotaessa

Viikolla 3 A:n yrittäessä vastata tehtävään (liite B2), joka edellytti oman koodin tuottamista, hänen ratkaisunsa eteni tarkalleen siten kuin Ristin teoria (käsitelty luvussa 1.5.1) ennustaa. Tämä siitä huolimatta, että A:n tuottamat ohjelmakoodin lauseet eivät olleet kokonaisia tai oikein.

A:n käsittelemässä tehtävässä A:ta pyydettiin tuottamaan ohjelma, joka aluksi kysyy käyttäjältä, montako lukua käyttäjä aikoo syöttää, ja sen jälkeen kysyy käyttäjältä halutun määrän lukuja yksitellen. A aloitti ohjelman työstämisen mainitsemalla lauseen, joka lukee käyttäjältä yhden luvun ja sijoittaa sen muuttujaan. Tämän jälkeen A tuli ajatelleeksi, että ennen luvun kysymistä käyttäjälle tulisi näyttää kehote luvun syöttämisestä. Tämän jälkeen A tuli ajatelleeksi, että tätä ennen käyttäjältä pitäisi kysyä syötettävien lukujen lukumäärä, niin kuin tehtävänannossakin sanottiin.

A:n käyttämä käsittelyjärjestys noudattaa Ristin mallin mukaista ”backward, bottom-up” -järjestystä, jota käytetään tilanteessa, jossa tarvittavat skeemat puuttuvat. Backward-osaa ilmentää se, että A tuotti ensin luvun lukemisen käyttäjältä ja vasta sitten kehotteen, joka ensin näytetään käyttäjälle. Tässä luvun lukeminen käyttäjältä on Ristin mallin mukainen laskentaosa ja kehotteen näyttäminen mallin mukainen alustusosa. Yhteensä nämä molemmat toimet ovat osa skeemaa, joka koko tehtävän kannalta muodostaa laskentaosan. Bottom-up-järjestystä ilmentää se, että A aloitti toimintansa hyvin yksityiskohtaiselta

tasolta (yksittäisen luvun kysyminen) ja vasta sen jälkeen siirtyi tarkastelemaan ohjelman kokonaisuuden tasoa (lukujen kokonaismäärän kysyminen ennen yksittäisten lukujen kyselyä). Kokonaisuuden tasolla lukujen kokonaismäärän kysyminen olisi alustusosa.

Tämä havainto validoi ennen kaikkea sen, että Ristin malli on esitetty sellaisessa muodossa, että se käytännössä soveltuu hyvin ohjelmoijien toiminnan tarkasteluun. Tässä tarkastellun opiskelijan käyttäytyminen myös noudatti Ristin mallin mukaista ennustetta, mutta koska kyseessä on vain yksittäinen tapaus, evidenssiä ei voida pitää luotettavana.

4.3.3. Yhden arvon simulointistrategia

C:n viikon 2 haastattelusta analysoitiin vain simulointiepisodi, ei koko haastattelua. Käsitelty koodinäyte on liitteessä A2. Simulointiepisodissa on useita kohtia, jotka antavat yhdenmukaista evidenssiä C:n simulointiin käyttämästä kognitiivisesta strategiasta. Lyhyesti ilmaistuna strategia on, että C piti mielessään tasan yhtä ei-triviaalia muuttujan arvoa ja käytti tätä arvoa heti tarvitessaan arvoa jollekin muuttujalle. Tällöin hän ei kiinnittänyt huomiota siihen, *minkä* muuttujan arvon hän oli painanut mieleensä tai *minkä* muuttujan arvona mielessä olevaa arvoa käytettiin. Mielessä pidettävät arvot olivat nimenomaan ei-triviaaleja muuttujien arvoja, kun triviaaliksi muuttujan arvoksi käsitetään arvo, joka ensimmäisenä sijoitetaan kyseiselle muuttujalle. Triviaalit arvot on mahdollista lukea suoraan ohjelmakoodissa olevista sijoituslauseista, eikä niitä siksi ole tarpeen pitää muistissa. Evidenssinä strategian olemassaolon ja käytön puolesta on kaksi erityisen selvää tilannetta simuloinnissa, joissa C käytti viimeiseksi mainittua arvoa seuraavassa tilanteessa, jossa tarvittiin jotain muuttujan arvoa. Näissä tilanteissa C sekoitti, minkä muuttujien arvoista kulloinkin oli kyse. Tilanteet kuvataan tarkemmin alla.

Strategian käytöstä voidaan johtaa ennuste, että C ei simuloinnissaan selviäisi tilanteesta, jossa tulisi verrata toisiinsa kahta ei-triviaalia muuttujan arvoa. Tämä ennuste toteutui, sillä ensimmäisessä kohdassa, jossa tällaista vertailua tarvittiin, C:n simulointi keskeytyi ja C lausui: ”Pirulauta, nyt mä sekosin kokonaan.” Lisäksi evidenssinä strategian puolesta on, että myöhemmin simuloinnin aikana C reflektoi omaa käsitystään ja lausui: ”Täytyyhän toi olla 64 plus 1, koska tämä lauseke ei suoranaisesti tohon oo mitenkään sidoksissa, eli toi lukuhan ei oo mitenkään sidoksissa tohon laskuriin, ne on aivan eri asia käytännössä...”

Episodissa ainoana strategian käytön vastaisena evidenssinä on tilanne, jossa viimeisin mainittu arvo oli 64, mutta seuraavan kerran arvoa tarvittaessa C käyttää arvoa 63. Tämä kuitenkin vaikuttaa huolimattomuusvirheeltä, koska haastattelijan kysyessä ”Hetkinen, 63?” C välittömästi korjasi arvon 64:ksi.

Ensimmäinen mainitun strategian käyttöä osoittava tilanne tapahtui, kun C oli simuloinut ulomman for-silmukan ensimmäisen iteraation ja yritti aloittaa toista iteraatiokertaa. Ensimmäisellä iteraatiolla `laskuri`-muuttujan arvoksi oli jäänyt 64. Ensimmäisen iteraation lopuksi luku-muuttujan arvo jaettiin kahdella, jolloin uudeksi arvoksi tuli 32. C siirtyi lauseesta ”`luku /= 2`” lauseeseen ”`laskuri++`” ja sanoi, että `laskuri`-muuttujan uudeksi arvoksi tulee 33. Tässä C siis käytti viimeksi mainittua arvoa, 32, `laskuri`-muuttujan arvona, vaikka mainittu arvo 32 oli ollut `luku`-muuttujan arvo ja `laskuri`-muuttujan viimeisin mainittu arvo oli 64. Tässä yhteydessä C:n simuloinnissa tapahtui virhe, sillä hän unohti lauseiden välissä käsitellä sisemmän for-silmukan, mutta unohdus liittyy oikean suoritusjärjestyksen ymmärtämiseen, eikä ole ristiriidassa tässä esitetyn kognitiivisen strategian kanssa.

Toinen strategian käyttöä osoittava tilanne tapahtui myöhemmin haastattelussa, kun C yritti uudelleen simuloida samaa tilannetta kuin äsken kuvatussa tilanteessa. C totesi, että laskurin arvo kasvoi 64:ään saakka ja että seuraavaksi luku-muuttujan arvoksi laskettiin 32. Tällä kertaa C siirtyi lauseen ”`luku /= 2`” jälkeen tarkastelemaan sisemmän for-silmukan ehto-osaa ja totesi, että `k`-muuttujan arvo on 32. Tuolloin viimeisin mainittu arvo 32 oli taas `luku`-muuttujan arvo ja `k`-muuttujan viimeisin arvo oli 64. Tässäkin tilanteessa C teki virheen, sillä hänen olisi pitänyt ensin käsitellä sisemmän for-lauseen ensimmäinen osa, mutta tämäkin virhe liittyy suoritusjärjestyksen tuntemukseen, eikä ole ristiriidassa esitetyn kognitiivisen strategian kanssa.

Mainitut kaksi tilannetta kattavat kyseisestä episodin kohdasta yhtä poikkeusta lukuun ottamatta kaikki mahdolliset tilanteet, joissa tällainen sekaannus on mahdollista tapahtua. Muissa simulointiepisodin tilanteissa, joissa muuttujan arvoa käytetään, käytettävä arvo saadaan joko triviaalina eli voidaan lukea suoraan koodista, tai kahdella muuttujalla on sama arvo, jolloin arvojen sekoittaminen ei aiheuta sekaannusta. Poikkeuksena on tilanne, jossa iteroidaan sekä muuttuja k että muuttuja laskuri 0:sta 64:ään. Tuossa tilanteessa C mainitsee erikseen lauseet ”laskuri++” ja ”k++” ja ymmärtää ne kohdistuviksi eri muuttu-

jiin. Noiden kahden muuttujan kasvattamiset ja käytöt tapahtuvat ajallisesti lyhyen ajan sisällä, ja muuttujien roolit ovat tuon tilanteen aikana lähes identtiset, joten muistin käytön kannalta niiden ”sulautuminen” yhdeksi arvoksi lienee ymmärrettävää.

4.3.4. Simuloinnin abstraktiotason nostaminen

Haastateltava B osasi jo viikolla 1 käsitellä if-else-lauseen toiminnan ja vaikutuksen yhtenä kokonaisuutena. Lisäksi viikolla 1 hän ymmärsi for-lauseelta halutun funktion ja osasi käyttää tätä tietoa oman simulointinsa kriittiseen tarkasteluun. Viikolla 2 kaikki kolme haastateltavaa, joiden simulointiepisodit analysoitiin, osasivat jossakin vaiheessa simulointia nostaa oman simulointinsa abstraktiotasoa. Viikon 2 koodinäytteessä (liite A2) oli kaksi sisäkkäistä silmukkaa. Yhteistä kaikkien kolmen haastateltavan simuloinnille oli, että toistettuaan ulomman silmukan iteroinnin yhteen tai kahteen kertaan, he osasivat jatkossa simuloida yhden iteraatiokerran vaikutukset yhtenä kokonaisuutena, käymättä läpi sisemmän silmukan yksittäisiä iteraatioita. Vaikutusten käsitteleminen yhtenä kokonaisuutena osoitti, että he olivat abstrahoineet suorituksen yksittäiset vaiheet pois ja pystyivät toteuttamaan mentaalisen operaation, jossa useiden vaiheiden vaikutukset oli yhdistetty yhdeksi päättelyaskeleeksi.

Viikolla 3 haastateltava B simuloi samaan tyyliin hieman vaativampaa koodinäytettä (liitteet A3 ja A4), mutta ei enää pystynyt nostamaan simulointinsa abstraktiotasoa. Epäselväksi jäi, mikä tekijä viikon 3 haastattelun simulointitehtävässä oli sellainen, että B ei pystynyt automatisoimaan simulointiaan. Viikon koodinäytteessä toteutetaan kahden vierekkäisen alkion paikan vaihto taulukossa. B osasi itsekin sanoa, että lauseet toteuttavat paikanvaihdon, eli hän ymmärsi lauseiden funktion. Kun B oli jo simuloinut kaksi paikanvaihtoa, haastattelija kysyi, voisiko B luopua tarkastelemasta erään muuttujan (nimeltään apu) arvoa. Kyseistä muuttujaa käytetään ohjelmassa ainoastaan apumuuttujana taulukon alkioiden paikkaa vaihdettaessa, ei mihinkään muuhun. Kun paikanvaihto-operaation toiminta on tiedossa, paikanvaihdon vaikutukset olisi mahdollista käsitellä pitämättä erikseen kirjaa apu-muuttujan arvosta. B siis osoitti tuntevansa paikanvaihto-operaation toiminnan, mutta siitä huolimatta hän ei edes haastattelijan ehdotuksesta suostunut luopumaan apu-muuttujan kirjaamisesta erikseen. Käytiin seuraava keskustelu:

B (798): Sitten verrataan taas, mennään sinne while-lauseen alkuun, niin se edelleen, ehto pitää paikkansa. Sitten verrataan kolmos- ja nelospaikkoja. Ja taas pitää vaihtaa niiden paikat, koska kolmosen arvo on isompi kuin nelosen arvo, eli...
H (803): **Niinku sä iteki sanoit et niiden paikka vaihtuu, ni tarviiksä nyt, jos sä ajattelet sitä paikan vaihtoo, ni tarviiksä tätä apu-muuttujaa välttämättä?**
B (807): Siis ajattelussani, vai?
H (807): Niin, tai, tai tarvii... Jos sä niinku haluut vaan simuloida tätä toimintaa...
B (808): Siis jos mä haluan simuloida sitä niin että mä pysyn kokoajan ite kärryillä siitä mitä mä oon tekemässä, niin kyllä mä tarviin.
H (810): Joo. Okei.
B (810): Et mä... Tää on niin, on, kokoajan täytyy, sen takia mä puhun kaikki nää asiat myös ääneen, koska mun täytyy joka kerta ku mä nään näitä lauseita, ni miettiä et mitäs tämä nyt tarkottikaan.
H (813): Joo. Joo.
B (813): Niin tota ni, samalla lailla **mun täytyy kirjottaa tää apukin ylös, koska muuten mä tipahdan itse omasta simuloinnistani**.
H (815): Joo. Sä kuitenki huomasit heti ekan kerran ku sä teit sen operaation et siinon kyse niiden paikkojen vaihtamisesta.
B (818): Joo. En tiedä olisinko huomannut jossei tätä olis käyty läpi luennoilla aika monen esimerkin kautta.

B:n vastaus ilmentää, että hänen mielestään ohjelman suorituksen simulointi on vielä erittäin vaativaa. Muuttujien arvojen ylläpito ja päivittäminen onnistuivat häneltä simuloinnin aluksi moitteettomasti. Myöhemmin, tehtäessä paikanvaihtoja taulukossa, muuttujien arvojen ylläpito ja päivittäminen alkoivat käydä kognitiivisesti erittäin vaativiksi, ja B:llä tapahtui useita lipsahduksia ja huolimattomuusvirheitä.

Viikoilla 2 ja 3 käsiteltyjen koodinäytteiden vaativuudessa on selvä ero, mutta viikon aikana intensiivikurssi oli myös edennyt huomattavasti, sillä koko laajan kurssikokonaisuuden ajallinen kesto oli kuusi viikkoa. Viikon 2 koodinäytteessä (liite A2) on kaksi muuttujaa (`luku` ja `laskuri`), jotka pääasiassa määräävät ohjelman toiminnan, sekä yksi muuttuja (`k`), jota käytetään vain iteraatiokertojen laskentaan, ja yksi muuttuja (`l`), jota ei käytetä mihinkään. Viikon 3 koodinäytteessä (liitteet A3 ja A4) ohjelman suorituksen kulku määräytyy kahden muuttujan (`parhaillaan` ja `mihinAsti`) arvojen perusteella. Lisäksi viikon 3 koodinäytteessä manipuloidaan taulukon alkioiden arvoja sekä käytetään yhtä apumuuttujaa alkioiden paikan vaihtoon. Kummassakin koodinäytteessä on kaksi sisäkkäistä silmukkaa.

Samassa haastattelussa, kun metodin teeJotain simulointia oli käyty **ulomman silmukan ensimmäinen iteraatio loppuun**, haastattelija pyysi B:tä kuvaamaan käsitteellisellä tasolla, miten metodin suoritus jatkuisi:

> H (862): Joo. Osaisiksä nyt sanoo, että seuraavan kerran ku suoritetaan toi sisempi silmukka, näin käsitteellisesti, ni mitä sillon tehään?
> B (865): Sillon käydään, kun tän ensimmäisen silmukan, tai ensimmäisen kerran ku tää while-silmukka käytiin läpi, siis tää sisempi, niin saatiin siirrettyä isoin näistä arvoista, mitä tässä taulukossa on, viimeseks. Ja nyt käydään sitten kaikki ne, sitä viimestä lukuun ottamatta kaikki, muut viis arvoa. Samalla tavalla läpi ku mitä mä äsken selostin, jonka tuloksena sitten taas tästä jonosta isoin saadaan sen jonon viimeseks. Ja niin sitte edetään loppuun saakka. Ja lopputulos on sit se, että ne on suuruusjärjestykseen järjestetty ne taulukon arvot.

B:n esittämä päätelmä oli oikeansuuntainen, mutta ei aivan oikea. Ensinnäkin suurimman alkion saaminen viimeiseksi oli sattumaa. Suurin alkio oli valmiiksi toiseksi viimeisenä. Toiseksi lausuma jätti täysin huomiotta kaikki muut paikanvaihdot, mitä tehtiin. Lausuma ei siis kuvastanut kunnollista funktionaalista ymmärrystä käytetystä järjestämisalgoritmista. Luennolla oli käyty läpi vaihtojärjestäminen, kun taas tässä ohjelmassa oli käytössä kuplajärjestäminen. Vaihtojärjestämisessä todella saadaan ensimmäisellä iteraatiolla kaikkein pienin alkio taulukon ensimmäiseksi, mikä on kovin ”samanmuotoinen” ajatus kuin se, että saadaan suurin viimeiseksi. Se voisi selittää haastateltavan käyttämän väärän ”planin”. Virhekäsityksestä huolimatta B osasi kuvata metodin toimintaa kokonaisuutena aivan oikein, kohdassa:

> H (883): Joo. Okei. Eli nyt sä sait selville et mitä tää metodi tekee.
> B (884): Joo.
> H (885): Joo. Eli sille annetaan... Voiksä kertoo ite?
> B (886): Siis sille annetaan parametriks joku taulukko ja sitten se järjestää sen taulukon arvot, suuruusjärjestykseen pienimmästä suurimpaan.
> H (892): Joo. Ja se palauttaa...?
> B (893): Ja se palauttaa sen taulukon, järjestetyn taulukon arvonaan.

B:n simuloinnista viikolla 3 havaittiin myös, että tarkastellessaan pelkästään TekeeJotain-luokan teeJotain-metodin koodia, hän ymmärsi kyllä koodin syntaktisesti, mutta hän ei kyennyt päättelemään tai tarkastelemaan metodin toimintaa ilman todellista parametria. Sen sijaan myöhemmin, kun käytössä oli todellinen parametri (liitteessä A4 oleva koodinäyte oli esitetty liitteessä A3 olevan näytteen lisäksi), haastateltava osasi simuloida metodin suoritusta.

Haastatteluissa tuli siis jo ensimmäisen viikon haastatteluista lähtien usein esille tilanteita, joissa haastateltavat esittivät omia päätelmiä ohjelmakoodista ja siten onnistuivat nostamaan simulointinsa abstraktiotasoa. Haastatteluissa saatiin myös esimerkki siitä, miten

abstraktiotason nostaminen saattoi epäonnistua henkilöltä, jolta se oli jo aiemmin onnistunut.

4.3.5. Rakenteen ja toiminnan sekoittaminen

Luvussa 1.1.3 esiteltiin de Kleerin ja Brownin teoriaa suoritettavista mentaalisista malleista. Eräs siinä mainittu tärkeä periaate on no-function-in-structure-periaate, jonka mukaan robustissa mentaalisessa mallissa järjestelmän rakennetta koskeva tieto on erotettu järjestelmän toimintaa koskevasta tiedosta. Viikolla 2 haastateltavan C haastattelussa saatiin demonstraatio mainittua periaatetta rikkovasta mentaalisesta mallista. For-lausetta käytetään ohjelmissa usein niin, että alkuasetuksessa alustetaan muuttuja jollakin alkuarvolla (usein nollalla) ja muita osia käytetään tämän muuttujan askeltamiseen halutulla tavalla haluttuun arvoon asti. Jos mentaalinen malli perustuu tällaiseen mainittuun käyttötarkoitukseen ilman, että mallissa on selvästi erotettu toisistaan rakenne ja toiminta, saattaa tästä seurata virheellisiä oletuksia, kuten esimerkiksi että ”for-lauseen toistettavassa lauseessa käytetty muuttuja nollataan aina for-lauseen aluksi”. (Oikea malli for-silmukan rakenteesta olisi, että siinä on neljä osaa, kurssimateriaalin termein sanottuna alkuasetus, jatkamisehto, eteneminen sekä toistettava lause. Ohjelmointikieli määrittelee tietyn logiikan sille, missä tilanteissa mikäkin näistä osista suoritetaan, mutta ohjelmointikieli ei määrittele, että osien suorituksen olisi välttämättä sisällettävä jotakin tiettyä toimintaa.) Seuraavissa lainauksissa haastateltava C vaikuttaa soveltavan mainittua oletusta.

C oli simuloinut koodinäytteestä (liite A5) ulomman for-silmukan ensimmäisen iteraatiokerran ja oli aloittamassa seuraavaa kertaa. Haastattelija oli joutunut selittämään, että muuttujan k arvo ei ollut säilynyt edellisestä iteraatiokerrasta seuraavalle kerralle. Keskustelu jatkui seuraavasti:

> C (724): Just joo, elikä silloinhan se lähtee käytännössä kiertoon uudestaan.
> H (725): Paitsi että se ei oo k:n arvo mitä me tulostetaan.
> C (727): Ei vaan laskurin arvo lähtee kiertoon uudestaan. Jos k on nolla. Niin silloin taas ehto pitää paikkansa, elikä k on pienempi täs tapauksessa ku luku-muuttuja, elikä se on pienempi kuin 32.
> H (730): Mmm, nolla on pienempi kuin 32.
> C (731): Mmm, eli pitää paikkansa, silloin laskuri on tota, öö 64... hetkinen, mutta täähän tulee, onks se sillon 64 plus yks? Vieläkö tää laskuri...? **Ei mutta eiks sekin mee nollille?**
> H (736): Mmm.

C (736): Vai tuleeko tää tuolta mukaan tonne?
H (737): Nii. Tota. Mistäs me se voitais päätellä?
C (740): Sulkuja on niin pirusti... Tota...

Korostetussa kohdassa C esitti arvelun, että myös laskuri-muuttuja nollattaisiin. Haastattelu jatkui siten, että haastattelija yritti selventää epäselvyyttä muuttujien käyttäytymisessä, mutta siitä huolimatta C esitti saman arvelun uudelleen:

H (741): Jos mietitään, että se lohko, missä tää laskuri on alustettu, eli tää koko main-silmukka, ni se on alkanu täällä. Ja sehän loppuu vasta sit ku me päädytään tähän toiseks viimeseen kaarisulje kiinni -merkkiin. Mut sinne me ei oo vielä päädytty, eiks niin?
C (746): Aivan.
H (746): Eli se on ollu olemassa, koko ajan.
C (747): Okei.
H (747): Ja toisaalta täällon tää alustuslause, joka antaa, siis tää int laskuri = 0, niin, ni myöskään sitä lausetta, ni me ei oo siihen päädytty missään vaiheessa, eli silloin ku tullaan sisemmästä for-silmukasta ulos, käsitellään tätä ulompaa for-silmukkaa, ni **silloinhan me ei tehdä missään vaiheessa mitään muutoksia tohon laskurin arvoon. Eiks niin?**
C (755): **Aivan.**
H (755): Joo.
C (755): Elikä sen mukaan sen pitäis mennä sitte, tänne tulla yks, jos mä nyt luin sun jutuista oikein, koska ei me tänne, tänne me ei voida, siis tänne, tänne laskurin kohtaan. Vai luinko mä nyt väärin, mitä sä sanoit?
H (761): Hmm.
C (761): **Koska nyt edelleen for-lause alkaa uudestaan, ni kyllä mä näen että laskurikin on silloin nolla plus yks.**

Kohdan tarkastelussa on otettava huomioon, että ohjelman rakenteen hahmottaminen on C:lle vielä hankalaa. Lisäksi ennen lainauksia käyty keskustelu `k`-muuttujan arvon säilymisestä saattaa sekoittaa C:n ajattelua. Muuttujasta `k` siis keskusteltiin, että arvo ei säily ulomman silmukan iteraatiokerralta toiselle, vaan esitellään joka iteraatiolla uudelleen ja alustetaan nollaksi. Tällöin C:n saattaa olla hankala ymmärtää, miksi `k` alkaa aina nollasta, mutta `laskuri`-muuttuja ei.

Lainauksista kuitenkin näkyy, että C sitkeästi ja kahteen kertaan kuvittelee ohjelmaan toiminnon, joka for-lauseen suorituksen aluksi nollaisi for-lauseen sisällä käytetyn muuttujan. Ohjelmassa ei tällaista toimintoa ole. Tämä on esimerkki siitä, millaisia virhepäätelmiä saattaa syntyä, jos mekanististen mentaalisten mallien teorian yhteydessä esiteltyjä mentaalisten mallien robustiussääntöjä rikotaan.

Toinen esimerkki selvästi funktionaalista tietoa sisältävistä käsityksistä saatiin A:n viikon 1 haastattelussa.

(#180)
H: Entä tota mitä sun mielestä on syöte ja tuloste?
A: No siis syöte on se että sille tietokoneelle täytyy viedä se tieto semmosessa muodossa, että se tietokone ymmärtää sen. — —
H: Ajatellaan et me tehään joku ohjelma, ja sit me käynnistetään se ohjelma, niin mikä sen ohjelman kannalta on sitte syöte?
A: Siis ohjelman kannalta on syöte sellanen tieto mitä se ohjelma ei muuten voi tietää — —
H: Jos meillon käynnissä oleva ohjelma, niin syöte on sillon... Voiksä jatkaa? Asioita mitä se käyttäjä...?
A: Siis se syöte on semmonen asia, mitä se käyttäjä tietää.

Edellisestä lainauksesta huomataan, että A:n vastaukset eivät liikkuneet konkreettisella ohjelmien suorituksen tai ohjelmakoodin tasolla, vaan ne käsittelivät *funktionaalisia* seikkoja. Seuraava keskustelu käytiin käsiteltäessä määritelmiä eri termeille:

(#263)
H: Okei, entäs sitte, merkkijono?
A: Merkkijono?
H: Nii.
A: Elikkä siis oliks se tää String?
H: Joo.
A: No se on sellanen, millä tietokone ei operoi, se ei suorita sillä laskutoimituksia, eikä tee mitään muutakaan, se vaan käsittelee sen yhtenä kappaleena.

A:n määritelmä ei kerro mitään siitä, millainen on merkkijonon rakenne tai sisältö, määritelmä kertoo pelkästään yhden *funktionaalisen* seikan, joka liittyy merkkijonoilla operoimiseen. A:n lainauksissa vaikuttaa erikoiselta se, miten hänen vastauksensa liikkuvat varsin abstraktilla ja funktionaalisella tasolla. Vastaukset ovat ensimmäisen viikon haastatteluista ja kysymys saattaa olla myös opiskeluteknisestä ongelmasta, eli A:lle ei ollut selvinnyt, millainen tieto on olennaista ohjelmien kirjoittamisen ja ohjelmien toiminnan ymmärtämisen kannalta.

4.4. Ulkoisten apuvälineiden käyttöön liittyvät havainnot

4.4.1. Tietoesitysten käyttö

Tietoesityksellä tarkoitetaan tässä mitä tahansa ulkoista (vastakohtana mielensisäiselle) tietoesitystä, jota käytetään apuna päättelyssä. Tietoesitys voi olla esimerkiksi kuva tai kaa-

vio. Viikon 6 koodinäyte (liite A5) oli sellainen, että sopivaa tietoesitystä käyttämällä ohjelman suorituksen pystyisi simuloimaan hyvin vähällä vaivalla. Analysoiduista simulointiepisodeista tarkasteltiin, miten haastateltavat B ja C muodostivat ja käyttivät tietoesityksiä.

C ei omasta aloitteestaan piirtänyt tai sanonut mitään sellaista, jonka perusteella hän olisi muodostanut ohjelman tietorakenteista jonkinlaisen tietoesityksen omaan käyttöönsä. Hän etsi tarvitsemansa tiedot suoraan koodista. Kun haastattelija kehotti kirjoittamaan ohjelman toiminnasta jotakin paperille, C ilmaisi seuraavan ajatuksen: ”...mä mietin sitä että tota, ku ei tätä nyt voi piirtää näin että tää saa suoraan mitään arvoja, vaan ku tän käynnistää, ni tulee uus lista hip, joka saa niinkun...” C:n lausumasta syntyy spekulatiivinen vaikutelma, että yksittäisille paikallisille muuttujille C olisi osannut pitää kirjaa arvoista, mutta kun ohjelmassa oli useita saman luokan ilmentymiä, C ei keksinyt, miten olioiden kenttien arvoista olisi voinut pitää kirjaa.

Kun haastattelija pyysi C:tä piirtämään olioita ja niiden kenttien arvoja, C:n piirros oli rakenteeltaan sama kuin olioiden luontilauseet kirjoitettuna paperille. C sanoi, että ei hallitse olioiden piirtämistä, koska ei ollut aiemmin piirtänyt niitä. C:n tekstiesitykseen tuli kuitenkin mukaan olion kentän arvona oleva null-arvo, vaikka sitä ei koodissa esiinny luontilauseen parametrina. C ”piirsi” seuraavat rivit tekstiä:

```
Lista hip =
hip ("hip , null)
hap ("hap",
```

Näistä keskimmäinen rivi oli ympäröity laatikoksi. Viimeinen rivi oli selvästi jäänyt kesken. Kesken jäänyt kohta kuvastaa, että C ei osannut esittää olioviitettä piirretyssä kuvauksessa, vaikka esimerkkejä sellaisista piirroksista oli kurssin luennoilla käytetty runsaasti.

Samalla viikolla toisen haastateltavan, B:n, haastattelussa haastattelija pyysi B:tä kirjaamaan paperille ”olioita tai muuttujia”. B kirjasi allekkain olioiden `sisältö`-kentän arvoja ja viereiseen sarakkeeseen `seuraava`-kentän arvoja. (Muuttuja `sisältö` oli merkkijonotyyppinen ja muuttuja `seuraava` viitetyyppinen.) Hip-olion kohdalle B oli kirjannut null-arvon, mutta kirjaamiensa hap- ja huu-olioiden `seuraava`-kentän kohdan B oli jättänyt

tyhjäksi. Haastattelijan kysyessä tästä B vastasi jättäneensä kohdat tyhjäksi, koska ei tiennyt, mitä niihin voisi kirjata.

Jäljempänä haastattelussa haastattelija pyysi B:tä piirtämään laatikko- ja nuoliesityksen ohjelman olioista ja niiden välisistä viittauksista:

> H (659): Saisinks mä ehdottaa sellaista, et jos nää oliot, mitä täs luodaan, ni jos ne vois jonkunlaisina, vaikka laatikoina, piirtää, niin et sinne laatikon sisälle tulis niiden olion kenttien arvot, ja sitten ku tässon tää Lista-tyyppinen muuttuja seuraava, ni jos sen arvon vois piirtää ihan tollasena nuolena, et ku se viittaa johonki toiseen olioon, ni piirtäis sen sellasena konkreettisena nuolena.

B suoriutui tästä piirtämistehtävästä helposti, mutta totesi, että piirtäminen ei selkiyttänyt hänen ajatteluaan lainkaan. B:n esittäessä simuloinnin loppuvaiheessa oikeita päätelmiä koodinäytteen toiminnasta, haastattelija kysyi, käyttikö B kaavioesitystä apuna päätelmiä tehdessään. B vastasi, että ei käyttänyt, vaan katsoi tarvitsemansa tiedot koodista.

5. Pohdinta

5.1. Menetelmän pohdinta

5.1.1. Haastattelumenetelmän käyttö

Tämän tutkimuksen tarkoitus oli eksploratiivisesti dokumentoida haastateltujen opiskelijoiden mentaalisia malleja, jotka liittyivät ohjelmointiin. Tutkimuksessa ei siis testattu jotakin aiemmin esitettyä hypoteesia, vaan pyrittiin tuottamaan tietoa, jonka avulla olisi mahdollista muodostaa jatkotutkimusten pohjaksi kokonaan uusia hypoteeseja, joita ei ole aiemmin esitetty. Menetelmän käytön onnistuneisuutta tarkastellaan tätä tarkoitusta vasten.

Haastatteluissa käytetyt tieto- ja määrittelykysymykset edellyttivät vastaukseksi sopivan sanallisen määritelmän tuottamista tai muistamista. Määrittelykysymyksiin vastattaessa periaatteessa olisi ollut mahdollista toistaa ulkoa opittuja määritelmiä ymmärtämättä niitä. Tätä tutkimusta varten tehtyjen haastattelujen luonne oli varsin vuorovaikutteinen ja haas-

tattelijan tietoisena tarkoituksena oli päästä sanallisten määritelmien pintatasoa syvemmälle, nimenomaan havainnoimaan haastateltavien mentaalisia malleja. Jos siis haastattelutilanteessa haastateltavan vastaus vaikutti ulkoa opitulta määritelmältä, haastattelija esitti lisäkysymyksiä, joihin vastaaminen edellytti ymmärrystä määriteltävästä käsitteestä ja sen suhteesta muihin käsitteisiin. Tällaisissa tilanteissa haastattelujen joustava ja vuorovaikutteinen rakenne palveli hyvin mentaalisten mallien havainnoinnin tavoitetta.

Niissä haastattelutilanteissa, joissa haastateltava työskenteli ohjelmakoodin parissa, käytetty menetelmä muistutti hieman ääneenajattelumenetelmää (esimerkiksi Ericsson & Simon, 1984). Näissäkin tilanteissa haastattelut olivat varsin vuorovaikutteisia siten, että haastattelija välillä suuressakin määrin ohjasi haastateltavan työskentelyä. Esimerkkinä tällaisesta ohjaamisesta on tilanne, jossa haastattelija esittää kysymyksen ”Mikä on seuraava suoritettava lause?” sen sijaan, että olisi antanut haastateltavan edetä spontaanisti. Vaihtoehto tällaiselle vuorovaikutteisuudelle olisi, että että haastattelijan kysymykset olisivat vain kehotuksia puhua ääneen, eivätkä ohjaisi haastateltavan työskentelyä, esimerkiksi: ”Mitä ajattelet nyt?” Jos tarkoitus olisi havainnoida koehenkilön kognitiivista prosessia itsenäisen työskentelyn aikana, haastattelijan puuttuminen haastateltavan työskentelyyn luonnollisesti vaikuttaisi haastateltavan kognitiiviseen prosessiin ja siten ”pilaisi” prosessia koskevat havainnot. Tässä tutkimuksessa kuitenkin haluttiin havainnoida mentaalisia malleja, eikä tutkittavaa tehtävätyyppiä rajattu itsenäiseen työskentelyyn. Mentaalinen malli eroaa menetelmän kannalta itsenäisen työskentelyn kognitiivisesta prosessista sikäli, että mentaalisista malleista voidaan tehdä päätelmiä käsityksiä kuvastavien lausumien perusteella, eikä tällaisten lausumien ole välttämätöntä sijoittua samaan häiriöttömään ajatuskulkuun samalla tavoin kuin kognitiivista prosessia havainnoitaessa. Päinvastoin, mahdollisuus esittää tarkempia kysymyksiä ja kysyä käsitysten perusteluja parantaa mahdollisuuksia tehdä havaintoja mentaalisista malleista.

Tässä tutkimuksessa tutkitut henkilöt oli nimenomaan valittu siten, että he eivät ennestään osanneet ohjelmointia, ja haastatteluja tehtiin aivan haastateltavien opiskelun alusta lähtien. Tällöin varsinkin ensimmäisissä haastatteluissa haastateltavien taitotaso oli sellainen, että oli varsin epätodennäköistä, että he olisivat itsenäisesti suoriutuneet ei-triviaaleista tehtävistä kohtalaisen lyhyessä ajassa. Siksi oli perusteltua järjestää haastattelut siten, että haastattelija tarvittaessa ohjasi haastateltavia heidän tehdessään tehtäviä.

Vaikka haastattelumenetelmää on laajasti käytetty ohjelmointia käsittelevissä tutkimuksissa, aiemmista tutkimuksista ei ollut löydettävissä mallia tai apua tässä tutkimuksessa tehtyyn haastatteluaineiston analyysiin. Erityisesti haastatteluissa käsiteltyjen aiheiden laaja-alaisuus ja eksploratiivinen tutkimusongelman asettelu saivat aikaan sen, että tämän tutkimuksen haastatteluaineiston analysointi oli haastavaa ja edellytti paljon itsenäistä työtä. Myös se, että tässä tutkimuksessa haastateltavia oli vähän, ja että samoja haastateltavia haastateltiin useaan kertaan, teki verrattain strukturoimattoman haastatteluaineiston käsittelyn melko hankalaksi. Hankaluus johtui siitä, että opiskelijoiden tiedot ja työtavat poikkesivat toisistaan niin paljon, että eri opiskelijoiden kanssa käydyt keskustelut olivat harvoin vertailukelpoisia keskenään.

Fleuryn (1991, 2000, 2001) ja Eckerdalin ja Thunén (2005) tutkimuksissa haastateltiin ohjelmointikurssin lopuksi kohtuullisen kokoinen otos (Eckerdal & Thuné 14, Fleury 23–28) opiskelijoita siten, että jokaista opiskelijaa haastateltiin yhden kerran. Näissä tutkimuksissa haastattelujen aihealue oli melko rajattu, joten pystyttiin tarkastelemaan eri opiskelijoiden esittämien käsitysten variaatiota samasta asiasta. Tästä huolimatta Eckerdalin ja Thunén (2005) tulokset, jotka esiteltiin luvussa 1.5.4, ovat suppeat, eivätkä juurikaan poikkea siitä, mitä analysoituja käsitteitä koskevan normatiivisen tiedon perusteella olisi mahdollista päätellä. On suorastaan mahdollista, että Eckerdalin ja Thunén tulokset toistavat suoraan asioita heidän analysoimansa kurssin oppimateriaalista tarjoamatta siihen mitään uutta näkökantaa. Verrattaessa tätä tutkimusta Fleuryn tutkimuksiin voidaan kuitenkin ajatella, että vaikka selkeämmin strukturoitu muoto olisi tehnyt analyysistä selkeämmän, se ei ehkä olisi ollut mahdollista niin, että kuitenkin säilytetään selkeästi eksploratiivinen tutkimusote. Toisaalta verrattaessa Eckerdalin ja Thunén tutkimuksiin voidaan ajatella, että haastattelun aihetta rajaamalla ja koehenkilöjoukkoa kasvattamalla ei välttämättä olisi saatu rikkaampia tuloksia kuin nyt.

5.1.2. Kirjallisten artefaktien käyttö

Kirjallisten artefaktien keräämistä ja analysointia on käytetty muissakin noviisien ohjelmointiin liittyvissä tutkimuksissa (esimerkiksi Lister *et al.*, 2004; Thomas *et al.*, 2004; Hundhausen & Brown, 2005). Piirtäminen on paljon käytetty keino esimerkiksi ohjelman

käsittelemien tietorakenteiden kommunikoinnissa. Tässä tutkimuksessa tutkitulla kurssilla luennoija käytti runsaasti piirtämistä havainnollistamiskeinona luennoidessaan.

Tässä tutkimuksessa koehenkilöiden piirtämillä ja kirjoittamilla asioilla oli merkitystä ainakin seuraavanlaisissa tilanteissa:

- Haastattelija pyysi haastateltavalta esimerkkiä tietynlaisesta ohjelmointikielen lauseesta. Tällöin vastaus on luontevaa antaa kirjoittamalla.
- Haastattelija pyysi haastateltavaa tekemään merkintöjä koodinäytteen yhteyteen, esimerkiksi merkitsemään toisiaan vastaavat aaltosuljeparit.
- Haastateltava piti kirjaa muuttujien arvoista simuloidessaan ohjelmakoodin suoritusta.

Näissä tilanteissa kirjallisten merkintöjen käyttö selkiytti kommunikointia haastattelussa. Lisäksi kirjanpito muuttujien arvoista luultavasti helpotti haastateltavien työmuistin kuormitusta ohjelmia simuloitaessa. Koska kirjalliset artefaktit olivat tallessa analyysivaiheessa, ne tukivat haastattelujen analysointia. Ilman näitä kirjallisia artefakteja joidenkin tapahtumien selvittäminen haastatteluista olisi saattanut olla hankalaa. Esimerkiksi joissakin tilanteissa työskennellessään haastateltava ei aina lausunut ääneen käsittelemiään muuttujan arvoja, vaan kirjoitti ne paperille. Tällaisissa tilanteissa haastattelija luki arvot ääneen, jotta tapahtumien ajallinen järjestys oli jälkikäteen mahdollista selvittää ääninauhoilta.

Nyt käytetyssä menetelmässä tapahtumien kulun selvittäminen jälkikäteen onnistui ilman vaikeuksia. Muita vaihtoehtoisia haastattelujärjestelyjä olisivat voineet olla haastattelujen videointi tai jonkinlainen tietokoneavusteinen järjestely, jossa haastateltava olisi työskennellyt tietokoneella ja tietokone olisi tallentanut haastateltavan toimet kellonajan mukaan. Nämä vaihtoehdot olisivat kuitenkin olleet teknisiltä järjestelyiltään vaativampia kuin pelkän ääninauhan käyttö, eikä ääninauhan käyttöön liittynyt sellaisia ongelmia, joiden ratkaisemiseksi olisi ollut syytä ryhtyä monimutkaisempiin järjestelyihin.

5.1.3. Käytetty analyysimenetelmä

Haastattelumenetelmän heikkoudeksi mainitaan (Hirsjärvi & Hurme, 2000, s. 35), että haastattelumenetelmä sisältää paljon virhelähteitä ja vapaamuotoisen aineiston analysointi, tulkinta ja raportointi ovat usein ongelmallisia, koska siihen ei ole tarjolla valmiita malleja. Yleisiä virhelähteitä ovat aineiston käsittely (esimerkiksi virheet litteroinnissa) ja tutkijan tekemät tulkinnat aineistosta. Hirsjärvi ja Hurme toteavat (s. 189): ”Tutkijan on pystyttävä dokumentoimaan, *miten* hän on päätynyt luokittamaan ja kuvaamaan tutkittavien maailmaa juuri niin kuin hän on sen tehnyt.” sekä ”...kyse on tutkijan tulkinnoista, hänen käsitteistöstään, johon tutkittavien käsityksiä yritetään sovittaa.” Lisäksi mahdollinen virhelähde on, että kaikkea käytettävissä olevaa aineistoa ei ole otettu huomioon päätelmiä tehtäessä.

Tässä tutkimuksessa pystyttiin välttämään mainittuja virhelähteitä seuraavasti. Äänitteiden tekninen laatu oli hyvä ja litterointi onnistui ilman ongelmia. Tutkijan tekemä luokittelu ja ”tutkittavien maailman kuvaaminen” ovat tässä tutkimuksessa vain vähäisiä virhelähteitä, koska toisin kuin monissa muissa laadullisissa tutkimuksissa, tässä haastattelut keskittyivät vain rajattuun aihealueeseen, joka oli luonteeltaan tekninen. Kurssikokonaisuus, jolle haastateltavat osallistuivat opiskelijoina ja haastattelija opettajana, tarjosi valmiit käsitteet ilmiöistä puhumiselle.

Tässä kuvatun analyysiprosessin ymmärtämisen kannalta on tärkeä muistaa, että tutkimuksen kohteena oli nimenomaan ohjelmointikielen toimintaa koskeva ymmärrys. Ohjelmointikieli itsessään on formaali kieli, joka koostuu tietyistä rakenteista. Kielen formaaliin määrittelyyn sisältyy, että kielelle on olemassa yksiselitteinen määrittely siitä, miten kielellä kirjoitettuja ohjelmia suoritetaan. (Yksiselitteisyys ei ulotu kaikkiin teknisiin yksityiskohtiin, mutta tällä ei ole vaikutusta tässä käsiteltyjen ohjelmien yhteydessä.) Tarkasteltaessa opiskelijoiden ymmärrystä, kielen määrittelyssä käytetyt rakenteet ja käsitteet muodostavat valmiin pohjan ja jäsennyksen myös kieltä koskevan ymmärryksen tarkastelulle. Tämän vuoksi toisin kuin monissa laadullisissa tutkimuksissa, tässä tutkimuksessa tutkija ei joutunut alusta alkaen luomaan luokkia ja käsitteitä tutkimansa ilmiön kuvaamiseksi. Lisäksi ohjelmointikielen määrittely toimii objektiivisena mittana sille, onko ohjelman suoritusta koskeva ymmärrys oikeaa vai väärää. Tätä mittaa haastattelija käytti hyväkseen jo haastattelujen aikana siten, että hän syvensi käsitystä niistä asioista, joista haastateltava

osoitti virheellistä ymmärrystä, ja jätti vähemmälle huomiolle asiat, joista haastateltava osoitti oikeaa ymmärrystä. Tutkimusongelman mukaisesti haluttu tulos ei siis ole pelkästään tieto siitä, onko opiskelijan jotakin asiaa koskeva ymmärrys oikein vai väärin. Tutkimusongelman mukaisesti haluttiin dokumentoida mentaalisia malleja, mikä tarkoittaa ymmärryksen *rakenteen* dokumentointia. Tuloksia raportoitaessa virheellisen ymmärryksen raportoiminen on tarpeen, koska tällöin tulee raportoida virheellisen mentaalisen mallin *rakenne*, kun taas oikeasta ymmärryksestä riittää todeta, että ymmärrys oli oikeaa. Tällöin voidaan olettaa, että ilmiötä koskeva ymmärrys vastaa normatiivisesti oikeaa tietoa.

5.1.4. Tulosten luotettavuus

Jos analyysimenetelmä perustuisi aineiston segmenttien luokitteluun, yksi tapa varmistaa analyysin luotettavuus olisi käyttää toista tutkijaa apuna siten, että toinen tutkija luokittelisi käytetyn luokittelun mukaan ainakin osan aineistosta, ja kahden luokittelun vastaavuutta verrattaisiin. Tästä saataisiin laskettua luokittelijoiden välinen luotettavuus (inter-rater reliability) prosenttilukuna, joka antaisi käsityksen tehdyn luokittelun luotettavuudesta. Tällöin voitaisiin todeta, ylittääkö luotettavuus jonkin hyväksyttävän raja-arvon, esimerkiksi 90 %. Tässä tutkimuksessa tällainen luotettavuuden varmistaminen ei ollut sellaisenaan mahdollista, koska aineistolle ei tehty segmenttien mukaista luokitusta. Tässä tutkimuksessa aineistoa ryhmiteltiin havaintokokonaisuuksiksi sellaisen päättelyprosessin tuloksena, jossa tutkija käytti hyväkseen ohjelmointikielten tuntemusta ja tutkitun kurssin oppimateriaalin tuntemusta ja pyrki huomaamaan yhteyksiä aineiston osien välillä. Aineiston osien liittyminen samaan havaintokokonaisuuteen on perusteltu tulosluvussa kunkin kokonaisuuden yhteydessä erikseen. Havaintokokonaisuuksien muodostaminen edellytti niin paljon päättelyä, että ei olisi kohtuullista odottaa, että toinen tutkija löytäisi aineistosta kaikki samat havaintokokonaisuudet. Olisi mahdollista, että osa nykyisistä jäisi löytymättä, ja olisi mahdollista että löytyisi kokonaisuuksia, joita nyt ei löydetty. Tämän tutkimuksen tuloksista ei siis väitetä, että saadut tulokset kattavat kaikki tulokset, mitä analysoidusta aineistosta olisi mahdollista saada.

Toinen tapa tarkastella tämän tutkimuksen tulosten luotettavuutta on pohtia, hyväksyisikö toinen tutkija tässä tehdyt päätelmät, jos hänellä olisi käytössään sama aineisto ja ne *perustelut*, mitä nyt esitetään aineiston liittymisestä esitettyihin havaintokokonaisuuksiin. Osit-

tain ongelman ratkaisee se, että tarvittavat perustelut ja käytetyt aineiston kohdat tehdään tulosluvussa lukijalle näkyväksi, joten lukija voi seurata tehtyä päättelyä. Tällaisen tarkastelun antama kuva on rajallinen siinä mielessä, että aineistoa on valikoitu tulosluvussa esittämistä varten. Olisi mahdollista, että esitettyjen lainausten ulkopuolelle jää sellaisia tietoja, jotka antaisivat aiheen muuttaa tehtyjä päätelmiä. Tämän virheen mahdollisuutta yritettiin analyysivaiheessa välttää sillä, että havaintokokonaisuuksien muodostamisen jälkeen niissä lainatut haastattelut käytiin kokonaisuudessaan uudelleen läpi ja varmistettiin, ettei lainattujen kohtien ulkopuolelle jäänyt samaan aiheeseen liittyviä tietoja, jotka tulisi ottaa huomioon. Varmistuksen luotettavuutta voitaisiin lisätä toisen tutkijan käytöllä, mutta tehtävän edellyttämän perehtymisen ja työläyden takia tähän ei ryhdytty.

5.2. Tulosten pohdinta

5.2.1. Yliyleistäminen

Haastateltavan A käsitykset viikolla 3, jotka raportoitiin luvussa 4.3.1, oli suuressa määrin johdettu yhdestä ainoasta esimerkistä. Tästä seurasi, että A:n ymmärryksessä olion ja luokan käsitteet olivat sekaantuneet toisiinsa. Myös kirjallisuudessa on raportoitu saman virhekäsityksen mahdollisuus, vieläpä siten, että syyksi esitettiin sama syy kuin tässä tutkimuksessa (Holland *et al.*, 1997).

Muita esimerkkejä epäonnistuneista yleistyksistä saatiin tarkasteltaessa A:n ymmärrystä *tyypin* käsitteestä, mistä raportoitiin luvussa 4.2.1. A esitti käsityksen, että muuttujan tyyppi voi muuttua ohjelman suorituksen aikana. Perusteluksi tälle käsitykselle hän esitti kurssin harjoituksissa käsitellyn tilanteen, jossa yhden lausekkeen arvoa laskettaessa muuttujan arvo oli muunnettu tyypistä toiseen. A oli yleistänyt tämän *arvon* tyypin muuntamisen tarkoittamaan *muuttujan* (eli arvon säilytyspaikan) tyypin muuttumista.

Yllä mainitut tapaukset ovat esimerkkejä yliyleistämisestä. Niissä opiskelija oli tehnyt tietojensa ja havaintojensa perusteella liian laajan yleistyksen, joka ei enää pitänyt paikkaansa. Näitä havaintoja voidaan verrata tuloksiin, jotka Fleury sai haastatellessaan ohjel-

moinnin opiskelijoita alkeiskurssin jälkeen (Fleury, 2000). Myös Fleuryn tutkimalla kurssilla opetuskielenä oli ollut Java. Fleury raportoi haastattelujen perusteella virheellisiä ”sääntöjä”, jotka opiskelijat olivat muodostaneet ohjelmointikielen toiminnasta. Fleuryn raportoimiin virheellisiin sääntöihin päti, että jos säännöstä otetaan pois sana ”vain”, saadaan tosi väittämä. Esimerkki Fleuryn tutkimuksessa havaitusta opiskelijoiden muodostamasta säännöstä on (tämän tutkimuksen kirjoittajan suomentamana): ”Vain luvut tai lukuliteraalit ovat kelvollisia todellisia parametreja, kun muodollisen parametrin tyyppi on kokonaisluku.” Perusteluissaan tätä sääntöä käyttäneiltä opiskelijoilta oli jäänyt ymmärtämättä, että lukujen ja lukuliteraalien lisäksi kelvollinen parametri olisi mikä tahansa lauseke, jonka arvo on tyypiltään kokonaisluku. Fleuryn haastattelemat opiskelijat eivät siis olleet yliyleistäneet, vaan aliyleistäneet.

Pohdittaessa, millaisia esimerkkejä opetuksessa kannattaisi käyttää ja miten, voidaan ottaa huomioon myös ongelmanratkaisutaidon yleistämistä käsittelevä tutkimus. Yllä olevat tulokset liittyivät käsitteitä tai mekanismeja koskevan tiedon oppimiseen tietokoneohjelmointia opiskeltaessa. Catrambone on tutkinut tapoja parantaa opiskelijoiden kykyä yleistää oppimansa ratkaisutavat uusien tehtävien ratkaisemiseen (Catrambone, 1994, 1996). Catrambonen tutkimukset käsittelivät matemaattisten tehtävien ratkaisemista. Jos esimerkkejä opetettaessa tuodaan esiin esimerkkiratkaisujen välitavoitteet, oppijoiden on helpompi siirtää oppimaansa uusien tehtävien ratkaisemiseen, koska heidän on helpompi päätellä, minkä välitavoitteiden osalta tunnettua ratkaisua pitää muokata uutta tehtävää varten. Catrambone totesi myös, että merkittäessä ratkaisujen välitavoitteita esimerkkeihin väliotsikoilla, siirtovaikutuksen *(transfer)* voimakkuuteen *ei* vaikuttanut se, oliko väliotsikko merkityksellinen vai merkityksetön. Olennaista siis oli kiinnittää oppijan huomiota ratkaisun jakamiseen välitavoitteisiin. Catrambone arveli, että merkityksettömät väliotsikot pakottivat oppijat itse päättelemään välitavoitteen merkityksen ja tämä päättely taas edisti sellaisen ymmärryksen muodostamista, josta oli hyötyä uusia tehtäviä ratkaistaessa. Tämän vuoksi merkityksettömät väliotsikot saaneet koehenkilöt pärjäsivät siirtovaikutusta testaavissa tehtävissä yhtä hyvin kuin merkitykselliset väliotsikot saaneet.

Pohdittaessa ohjelmointikursseilla käytettäviä esimerkkejä voidaan olio-ohjelmoinnin käsitteiden oppimisen osalta ottaa huomioon Hollandin *et al.* (1997) antamat suositukset, joita käsitellään vielä lisää tämän tutkielman seuraavassa luvussa. Pohdittaessa esimerkkien muotoilua voitanee todeta, että opiskelijoiden huomiota olisi syytä kiinnittää siihen,

millaisia osatavoitteita esitetyt ratkaisut täyttävät. Ymmärtäessään esitettyjen ratkaisujen osatavoitteet, oppijat osaavat paremmin soveltaa ymmärrystään tilanteisiin, jotka vastaavat aiemmin opittua vain osittain.

5.2.2. Olion ja luokan käsitteet

Tuloksissa saatiin esimerkkejä olion käsitteeseen liittyvistä ongelmista. Tyypillistä esimerkeille oli, että opiskelijat osasivat lausua jonkinlaisia käsityksiä siitä, että oliolla on tietosisältöä, tai että olioiden määritelmään sisältyy myös ohjelman toiminnan määritteleminen jollakin tavalla, mutta nämä käsitykset eivät olleet selkeitä, eikä niillä osattu operoida täsmällisesti. C ei osannut viikolla 6 täsmällisesti ilmaista kyseessä olleen olion kenttien lukumäärää. Viikolla 3 B ei tuntenut lainkaan olion tilan käsitettä ja A sekoitti toisiinsa muuttujan ja olion käsitteet. Lisäksi A:n käsityksissä pääohjelman käsitteeseen oli liitetty ominaisuuksia, jotka kuuluisivat luokan käsitteelle. Viikolta 4 analysoiduissa C:n käsityksissä luokan käsite oli hyvin eriytymätön, hän mainitsi luokan yläkäsitteeksi, joka saattaa sisältää esimerkiksi metodeja ja olioita. Olion käsitettä selittäessään C ei maininnut lainkaan olion tietosisältöä, kuten ei myöskään olion tilaa selittäessään.

Tässä tutkimuksessa saaduista tuloksista olion ja muuttujan käsitteiden sekoittaminen vastaa hyvin sitä, mitä Holland *et al.* (1997) asiasta esittivät. Vastaavuus näkyy myös siinä, mitä tämän virhekäsityksen *syyksi* esitetään. Holland *et al.* esittivät, että tämä virhekäsitys johtuu sellaisista esimerkeistä, joissa oliolla on vain yksi esiintymämuuttuja. Luvussa 4.3.1 raportoitiin todiste siitä, että A:n käsitys juonsi juurensa kurssilla käytetystä esimerkistä, jossa nimenomaan oliolla oli yksi ilmentymämuuttuja. Holland *et al.* eivät raportoineet empiiristä evidenssiä esittämiensä käsitysten tueksi, mutta luvussa 4.3.1 raportoitua tulosta voidaan pitää konkreettisena evidenssinä siitä, että muuttujan ja olion käsitteiden sekoittaminen voi kausaalisesti johtua esimerkistä, jossa oliolla on vain yksi ilmentymämuuttuja.

B:n käsitykset *olion* ja *luokan* käsitteistä viikon 3 haastattelussa vaikuttivat selviltä. Hän osasi käyttää molempia käsitteitä oikein puhuessaan esimerkiksi kapseloinnista tai näkyvyysmääreiden merkityksistä. Kuitenkaan B ei tuntenut *olion tilan* käsitettä. Voidaan ajatella, että olion tilan käsitteellä ei ole erityistä merkitystä muutoin kuin sellaisissa tapauksissa, joissa olion tila (kenttien arvot) todella vaikuttavat siihen, miten olio reagoi ulospäin

(esimerkiksi vastaa metodikutsuihin). Tämän perusteella kapseloinnin käsitteen selkeäkään ymmärtäminen ei vielä johdata ymmärtämään olion tilan käsitettä. Holland *et al.* (1997) esittävät, että jos olioista saaduissa esimerkeissä olion tilalla ei ole vaikutusta siihen, miten olio reagoi ulospäin, niin seurauksena on olion ymmärtäminen tietueeksi eli pelkäksi tiedon säilytyspaikaksi. Johtopäätöksenä on, että olion tilan käsitteen omaksumiseksi tarvitaan esimerkkejä, joissa olioilla on selkeitä tiloja.

Haastateltavalla A viikolla 3 ja haastateltavalla C viikolla 4 oli sekava käsitys luokan käsitteestä. A:lla tämä ilmeni sekaannuksena pääohjelman käsitteen kanssa, ja C kuvasi luokkaa hierarkian ylimmäksi osaksi, joka saattaa sisältää ainakin olioita ja metodeita. Luokan käsite on selvästi abstraktimpi kuin esimerkiksi muuttujan tai ehtolauseen käsite. Luokan käsitteen visualisoiminen tai demonstroiminen on selvästi vaikeampaa kuin esimerkiksi muuttujien tai ehtolauseiden kaltaisten yksinkertaisten ja mekaanisten käsitteiden havainnollistaminen. Tästä seuraa myös se, että opiskelijoiden voi olla vaikea hahmottaa luokan käsitettä ja omassa työskentelyssään esittää sitä mitenkään konkreettisesti, esimerkiksi yrittäessään simuloida ohjelman toimintaa. Luokan käsitteeseen, kuten muihinkin käsitteisiin, voi ajatella liittyvän konkreettisten operaatioiden tason ja käyttötarkoitusta käsittelevän funktionaalisen tason. Funktionaalisen tason tieto sisältää tietoja siitä, miksi ja miten luokan käsitettä käytetään. Luokan tapauksessa funktionaalinen tieto sisältää abstrakteja ohjelmistoteknisiä suunnitteluperiaatteita. Luokan tapauksessa konkreettisen tason tieto sisältäisi tietoa siitä, miten luokka esitetään tietokoneen muistissa (esimerkiksi Scott, 2000, s. 559) ja miten se konkreettisesti vaikuttaa ohjelman toimintaan. Tällaisen mallin esittäminen olisi *käsitteellisen mallin* esittämistä, jota käsiteltiin luvussa 1.6. Käsitteellisten mallien on todettu edistävän ongelmanratkaisutaidon oppimista siten, että ne parantavat opitun tiedon siirtovaikutusta uusiin tilanteisiin. Ohjelmoinnin tapauksessa siirtovaikutus on oppimisen kannalta erityisen keskeistä, koska varsinkin aloittelijoiden kohdalla jokainen uusi tehtävä on uusi tilanne.

Seuraavassa havainnollistetaan sitä, miten käytetyssä opetusmateriaalissa saattaa vallitsevana tiedon lajina olla funktionaalinen tieto. Kyseessä on lainaus siitä, miten luokan käsitettä esiteltiin tässä tarkastellun kurssin oppimateriaalissa (Wikla, 2004, luku 2.6, lihavoinnit ja kursivoinnit alkuperäisestä lähteestä):

> Luokan yksi tärkeä käyttötapa on seuraavanlainen:

> Luokka määrittelee jonkin toiminnallisuuden, ”koneen piirustuksen”. Määrittelyn perusteella voidaan sitten luoda luokan ilmentymiä, **olioita**, ”koneita”, jotka toteuttavat tuon toiminnallisuuden.
>
> Ajatus on, että ”koneen” toiminnan yksityiskohdat piilotetaan käyttäjältä, siis *ohjelmalta (ja ohjelmoijalta!)*, joka käyttää luokan ilmentymiä. ”Koneen” ohjelmoijalla ja käyttäjällä on yhteinen sopimus siitä, miten konetta käytetään: so. millaisia ”mittareita”, ”nappuloita”, ”vipuja” yms. koneessa on ja mitä ne tarkoittavat. Mutta muuten luokan ohjelmoijan ja luokkaa käyttävän ohjelman laatijan ei tarvitse (eikä pidä) tietää toistensa tekemisistä. Sama henkilö voi toki toimia molemmissa tehtävissä!
>
> **Huom:** Nuo ”mittarit”, ”nappulat” ja ”vivut” toteutetaan luokan julkisina (public) metodeina! (*Tämä on osatotuus!*)
>
> Tällaisia luokkia voidaan kutsua **abstrakteiksi tietotyypeiksi**: luokan ilmentymän, olion, käyttäjä tietää, miten oliota *käytetään*, mutta hän ei tiedä miten olio on *toteutettu.* Luokan toteuttaja ei puolestaan välttämättä tiedä, mihin luokkaa käytetään, hän vain toteuttaa sovitunlaisen välineen.
>
> *Tässä on kyseessä yksi menetelmä ”unohtaa järjestelmällisesti” yksityiskohtia, jotta voitaisiin hallita isompia ongelmia kuin oikeastaan osataan. Menettelyllä on myös vaikutuksia ohjelman siirrettävyyteen, luotettavuuteen ja olemassaolevien ohjelmien uudelleenkäyttöön.*
>
> Näin käytettyinä luokat ovat **tyyppejä** siinä kuin jo tutuksi tulleet int, double, boolean ja String. On siis mahdollista määritellä muuttujia, joiden tyyppi on jokin itse määritelty luokka; olioita voi sijoittaa tuollaisten muuttujien arvoksi, olioita voi välittää parametreina, arvon palauttava metodi voi palauttaa olion, jonka tyyppi on jokin oma luokka... *Jo edellä on käytetty String-tyyppisiä olioita eli String-luokan ilmentymiä tähän tapaan!*

Kuten lainauksen alussa sanotaankin, teksti esittelee luokan *käyttötapaa* eli funktionaalista tietoa, mutta ei sitä, mikä *on* luokka. Käsitettä selvennetään konkreettisen metaforan kautta ja kuvauksessa on mukana eri näkökulmia luokan käyttöön (käyttäjä ja toteuttaja). Lähimmäs operationaalista tasoa tullaan mainittaessa, että ”nappulat” ja ”vivut” toteutetaan julkisina metodeina. Lisäksi käsitettä selvennetään muiden abstraktien käsitteiden avulla. Lainauksen jälkeen oppimateriaali jatkuu koodiesimerkeillä siitä, millaisia jäseniä luokalle voidaan määritellä. Lainauksessa ei ole sellaista konkreettista operationaalisen tason tietoa, jonka perusteella opiskelija pystyisi muodostamaan luokan käsitteestä sellaisen mentaalisen mallin, jonka avulla ohjelmien toiminnan simulointi olisi mahdollista. Tarvittava konkreettinen tieto voisi liittyä siihen, miten luokkia ja toisaalta olioita esitetään tietokoneen muistissa. Toki tällaisen tiedon esittämiseen ei tuon kokoinen tekstikappale riittäisikään, mutta huomattavaa on, että käsitteen esittely ei sisältänyt operationaalisen tason tietoa käytännössä lainkaan.

Luvussa 1.5.4 esitellyssä Eckerdalin ja Thunén tutkimuksessa (Eckerdal & Thuné, 2005) tulokset esitettiin niin suppeasti ja yleisellä tasolla, että niitä on vaikea suhteuttaa tämän tutkimuksen tuloksiin. Esitetyissä lainauksissa on esiintynyt mainintoja, joiden perusteella tässä tutkimuksessa haastateltuja opiskelijoita voisi yrittää sijoittaa Eckerdalin ja Thunén esittämistä luokan käsitteeseen liittyvistä ymmärryksen kategorioista kahteen ensimmäiseen kategoriaan. Tällaisella sijoittamisella tuskin kuitenkaan on suurta merkitystä. Eckerdalin ja Thunén tutkimuksessa luokka-käsitteen ensimmäistä kategoriaa selvennetään lainauksella, joka on ote opiskelijan vastauksesta kysymykseen heidän haastattelussaan. Lainauksessa mainitaan pääohjelman käsite, jota on käsitelty myös tässä tutkimuksessa. Lainauksessa sanotaan näin (tämän tutkimuksen kirjoittajan suomentamana): ”Luokka on, luulen että luokka on pieni ohjelma. Niin minä sen näen. Pieni ohjelma koko suuren ohjelman sisällä. Jos iso ohjelma on *pääohjelma*, niin luokka on pieni ohjelma, joka tekee jotain tiettyjä asioita.” Näin siis myös Eckerdalin ja Thunén tutkimilla opiskelijoilla pääohjelman käsite oli saanut sellaista erityismerkitystä, jota sen ei tarvitsisi saada. Pääohjelma on varmasti ensimmäisiä ohjelman rakenteen osia, joita opiskelijat kohtaavat ohjelmointikurssin esimerkeissä. Koska suppeat esimerkit halutaan pitää pieninä ja yksinkertaisina, suuri osa ohjelmien logiikasta sijoitetaan pääohjelmaan. Opetettaessa ohjelmien toimintaa ensimmäistä kertaa, pääohjelman käsitettä on vaikea sivuuttaa, koska pääohjelma toimii aloituskohtana ohjelmien suoritukselle. Kuitenkin jos tarkastellaan edistyneempää ymmärrystä olio-ohjelmoinnista, niin juuri tämä aloituskohtana toimiminen on pääohjelmametodin *ainoa* tehtävä, pääohjelma ei siis minkään muun ominaisuuden puolesta poikkea muista metodeista. Tämän asian esittäminen esimerkeillä kuitenkin edellyttää sellaisia esimerkkejä, joiden ymmärtämiseen tarvitaan jo melko edistynyttä ymmärrystä olioista ja metodeista, joten ainakaan ohjelmointikurssin alkuvaiheessa tällaisia esimerkkejä ei olisi mahdollista käyttää. Tästä huolimatta opiskelijoille voisi olla syytä korostaa sitä seikkaa, että pääohjelman ainoa erityismerkitys on toimia ohjelman suorituksen aloituskohtana. Näin voitaisiin vähentää riskiä siitä, että puutteellisten esimerkkien ansiosta pääohjelman ja luokan käsitteet sekaantuvat toisiinsa.

Olion käsitettä tutkittiin tässä tutkimuksessa haastateltavien A ja B viikon 3 haastatteluista, jotka litteroitiin kokonaan, sekä haastateltavan C viikkojen 4 ja 6 haastatteluista. C:n viikon 4 haastattelu analysoitiin, koska C:n lausumat viikolta 6 herättivät kysymyksen siitä, millainen ymmärrys C:llä oli olioista, ja viikon 4 haastattelussa oli kysytty olion, luokan ja olion tilan käsitteitä. Haastattelusarjassa olion ja luokan käsitteitä kysyttiin kaikissa haas-

tatteluissa viikosta 2 lähtien, mutta olion tilan käsitettä kysyttiin vain viikolla 4. Tällöin jos olioihin ja luokkiin liittyvistä käsityksistä olisi haluttu saada kattavammin tietoa, olisi ollut mahdollista tarkastella haastateltavien vastauksia suurimmasta osasta tehtyjä haastatteluja. Tähän ei kuitenkaan ryhdytty, koska tutkimuksen laajuutta ja työmäärää oli syytä rajoittaa.

5.2.3. Tyypin käsite

Toisin kuin olion ja luokan käsitteitä, tyypin käsitteen oppimisen ongelmaa ei ole aiemmin käsitelty ohjelmoinnin psykologian kirjallisuudessa. Tyypin käsite on kuitenkin eräs keskeisimmistä omaksuttavista käsitteistä ohjelmointia opiskeltaessa. Kaikilla ohjelmissa esiintyvillä muuttujilla ja arvoilla on aina tyyppi. Esimerkiksi Java-kielessä tyypin käsitettä monimutkaistaa sen kahtiajakoisuus, osa mahdollisista tyypeistä on alkeistyyppejä ja osa viitetyyppejä. Alkeistyypit käsitellään esimerkiksi parametrinvälityksessä arvosemantiikan mukaan ja viitetyypit viitesemantiikan mukaan, eli laadullisesti eri tavoin. Kukin luokka on samalla tyyppi, ja luokkien perintähierarkia muodostaa samalla tyyppien hierarkian, jossa polymorfisesti kutakin tyyppiä voi edustaa joko saman luokan olio tai jotakin luokan perinyttä aliluokkaa oleva olio. Ohjelmien suorituksessa tehdään monessa kohtaa implisiittisiä tyyppimuunnoksia, esimerkiksi tehtäessä sijoituksia tai käytettäessä arvoja. Nämä erittäin tiiviisti kuvatut tyypin käsitteeseen liittyvät seikat ovat kaikki ohjelmien ymmärtämiseen liittyviä perusasioita, ja kuitenkin suuri osa näistä asioista on erittäin abstrakteja, ja siksi asioiden havainnollistaminen opiskelijoille on haastavaa.

Luvussa 4.2.1 esiintyneistä A:n käsityksistä nähtiin, että viikolla 1 A esitti tyypin ehdollisena (”siis jos se muuttuja on määritelty, niin sit se voi saada tietyn tyyppisiä arvoja”) ja viikolla 3 hän esitti käsityksen, jonka perusteella sekä muuttujat että metodit liittyvät ”tiedon tyypin määrittelyyn”. Näihin käsityksiin saattaa olla syynä käytetyn terminologian aiheuttamat mielikuvat ja se, että henkilö ei tuntenut termien teknistä merkitystä ohjelmoinnin yhteydessä. Sekä ”tyyppi” että ”määritteleminen” ovat sanoja, jotka esiintyvät samankaltaisissa merkityksissä myös yleiskielessä.

Viikolla 1 A:n esittämä käsitys, että muuttujan tyyppi voisi muuttua, johtui siitä, että A oli virheellisesti yleistänyt arvoa käytettäessä tehdyn implisiittisen tyyppimuunnoksen koskemaan myös sitä, että muuttujan tyyppi olisi muuttunut. Tässä tilanteessa A:lla ei ollut

oikeaa käsitteellistä mallia siitä, mitä tarkoittaa muuttujan käyttäminen tai lausekkeen arvon laskeminen. Nämä ovat hyvin perustavanlaatuisia operaatioita ohjelmien toiminnassa. Voidaan esittää kysymys, voisiko tällaisten asioiden oppimista tehostaa esimerkiksi sopivalla havainnollistamistavalla. Luvussa 1.6 mainittu Jeliot-havainnollistamisohjelma (esimerkiksi Levy *et al.*, 2003) animoi lausekkeiden arvon laskemisen vaihe vaiheelta ja esittää muuttujan arvon käyttämisen visuaalisesti omana operaationaan. Tällaisten animointien hyöty ei kuitenkaan ole lainkaan kiistaton, mitä asiaa tarkastellaan lisää seuraavassa luvussa.

Luvussa 4.2.4 B osasi määritellä, mikä on *oliotyyppinen muuttuja*, mutta esitti määritelmän vain tyypin kannalta eli niin, että tyyppi rajoittaa sitä, minkä olion muuttuja voi saada arvokseen. B tiesi int-tyypin olevan alkeistyyppi, mutta ei osannut selittää, miten alkeistyypit eroavat muista tyypeistä. Kummankaan käsitteen määrittelyn yhteydessä B ei maininnut olioviitteistä mitään. Jotta voisi selkeästi ymmärtää kahtiajaon alkeistyyppien ja viitetyyppien välillä, tulee ymmärtää olioviitteen käsite, sillä viitteiden käyttö tai käyttämättömyys on nimenomaan asia, joka erottaa alkeistyypit muista tyypeistä.

Tyypin käsitteen opettamisessa saattaa olla problemaattista se, että perinteisesti kokonaisluvut ja merkkijonot ovat selkeitä ja intuitiivisia asioita, joiden varaan ensimmäiset esimerkit on totuttu rakentamaan. Tällöin heti alusta pitäen esimerkeissä esiintyy sekä arvosemantiikkaa että viitesemantiikkaa, mutta eron selkeä selittäminen edellyttäisi huomattavasti abstraktimpien käsitteiden käyttöä. Eräässä yli 500 opiskelijaa eri oppilaitoksista kattaneessa tutkimuksessa havaittiin, että opiskelijat pitivät juuri viitteitä vaikeimpana yksittäisenä asiana ohjelmointikielten rakenteena oppimisen kannalta (Lahtinen *et al.*, 2005).

5.2.4. Tietoesitysten käyttö

Viikon 6 koodinäytteen käsittelyssä osoittautui, että pyydettäessä C ei osannut kuvallisesti esittää useasta oliosta koostuvaa linkitettyä tietorakennetta. Haastateltava B osasi, mutta sanoi itse, että ei käyttänyt piirtämäänsä kaaviota päättelyssään, vaan katsoi tarvitsemansa tiedot suoraan koodista.

On luonnollista, että B teki päätelmänsä tavalla, joka hänelle oli helpoin. Tuloksesta voidaan siis päätellä, että B:lle oli helpompaa katsoa tarvitsemansa tiedot ohjelmakoodista kuin itse piirtämästään kaaviosta. B oli piirtänyt kaavion omakätisesti, mutta haastattelijan pyynnöstä ja haastattelijan ohjeiden mukaan. Vaikka B onnistuikin vaivatta kaavion piirtämisessä ei ole selvää, että hänen apperseptionsa (käsitelty luvussa 1.6) kaaviosta onnistuisi ongelmitta. B siis pystyi helpommin muodostamaan vastaavuuden ohjelmakoodin lauseiden ja käsitteellisen tiedon välille kuin kaavion osien ja käsitteellisen tiedon välille. Tästä voidaan johtaa päätelmä, että tutkittaessa erilaisten tietoesitysten käyttöä opetuksessa, tarkasteltavia tai kontrolloitavia tekijöitä eivät ole pelkästään ne, näkevätkö tai tuottavatko opiskelijat tietoesityksiä, vaan nimenomaan se, pystyvätkö opiskelijat tekemään omia päätelmiä tietoesitysten pohjalta.

5.2.5. Simulointi ja päättely

Luvussa 4.1.1 nähtiin, että A:n simulointi epäonnistui siksi, että A ei tuntenut for-lauseen osien suoritusjärjestystä. A kuitenkin osoitti selvää yritystä koodinäytteen simulointiin, ja lisäksi hän osoitti tietävänsä, miten koodin toimintaa pystyttäisiin simuloimaan kynällä ja paperilla, pitäen kirjaa muuttujien arvoista. Samalla viikolla B osasi päätellä for-lauseelta halutun funktion, mutta hänkään ei onnistunut simuloimaan koodia, koska ei tuntenut for-lauseen osien suoritusjärjestystä. Nämä tulokset konkreettisesti vahvistavat sen seikan, että onnistuneeseen simulointiin tarvitaan riittävän täsmällistä operationaalisen tason tietoa ohjelmien toiminnasta. Funktionaalinen tieto tai ymmärrys simuloinnin käytännön toteutuksesta eivät riitä, jos täsmällinen ymmärrys ohjelman operaatioista puuttuu.

Luvussa 4.1.3 kuvattiin, miten viikolla 3 B yritti simuloida ohjelmakoodia ensin ”symbolisesti” eli ilman konkreettisia arvoja ja sitten ”konkreettisesti” eli siten, että käytössä oli konkreettiset arvot, joita ohjelma käsitteli. Détienne ja Soloway tunnistivat ”symboliseksi simuloinniksi” ja ”konkreettiseksi simuloinniksi” kutsumansa strategiat aineistosta, jossa oli haastateltu eksperttiohjelmoijia (Détienne & Soloway, 1990, käsitelty luvussa 1.4.2). Détienne ja Soloway kuvasivat, että symbolisessa simuloinnissa ohjelmoijat käyttivät runsaasti hyväkseen heillä olevaa tietoa ohjelmointiskeemoista ja ohjelmien roolirakenteesta. Symbolisen simuloinnin tavoite oli tunnistaa ohjelmasta ennestään tuttuja skeemoja. Tämän vuoksi on loogista, että aloittelija ei voi suoriutua symbolisesta simuloin-

nista, koska aloittelijalla ei ole käytettävissään valmista skeemoja koskevaa tietoa, jonka perusteella ohjelmia pystyttäisiin tulkitsemaan. Konkreettisessa simuloinnissa taas käytetään hyväksi ohjelman tietovuosta muodostettua dynaamista representaatiota, jonka yksinkertaisessa tapauksessa aloittelijakin pystyy muodostamaan. Tästä tuloksesta voitanee päätellä, että ohjelmoinnin peruskurssilla tulisi käsitellä vain sellaisia koodiesimerkkejä, joiden toimintaa opiskelijat pystyvät helposti simuloimaan konkreettisia arvoja käyttäen.

Viikolla 3 B:n simuloidessa koodia haastattelija kysyi, pystyisikö B yksinkertaistamaan simulointiaan. Tämä käsiteltiin luvussa 4.3.4. B sanoi perusteluna huolellisen simuloinnin tarpeelle, että hän joutuu huolellisesti miettimään jokaista koodin kohtaa pystyäkseen käsittelemään sen. Myös muut tilanteet B:n viikon 3 haastattelussa osoittivat, että vaikka B periaatteessa hallitsi ohjelmien syntaksin, syntaksin käsittely ei ollut automatisoitunut, vaan vaati huolellista käsittelyä ja lisäksi siinä tapahtui virheitä. Tästä herää kysymys, millä edellytyksillä simuloinnin abstraktiotason nostaminen sitten yleisesti olisi mahdollista. Tulosten perusteella eräs edellytys olisi kohtalaisen automatisoitunut syntaksin käsittely. Automatisoitunut syntaksin käsittely tarkoittaisi, että ohjelmaa lukiessaan henkilö pystyy ymmärtämään kunkin syntaktisen merkinnän merkityksen ilman tietoista muistelemista tai päättelyä. Toinen edellytys voisi olla kyky muodostaa ohjelmasta tehokkaita representaatioita. Edellä mainittu dynaaminen representaatio ohjelman tietovuosta voisi olla tällainen representaatio. Tieto ohjelman tietovuosta voisi mahdollistaa esimerkiksi sellaisen kvalitatiivisen päätelmän, että tietty muuttuja osallistuu vain kahden muuttujan paikkojen vaihtoon, eikä tämän muuttujan arvoa sen jälkeen käytetä mihinkään, joten arvoa ei tarvitse erikseen ylläpitää. Tällaisen päätelmän avulla B olisi pystynyt yksinkertaistamaan simulointiaan viikolla 3. Koska tulosten mukaan B kuitenkin viikolla 2 onnistui nostamaan simulointinsa abstraktiotasoa, mutta ei enää viikolla 3, voidaan todeta näiden taitojen olevan luultavasti sidoksissa käytettyihin ohjelmointikielen rakenteisiin. Kyse ei siis ole pelkästään oppijakohtaisesta kognitiivisesta taidosta, joka mahdollistaisi tietynlaiset päätelmät yhdenmukaisesti tilanteesta toiseen. Eräs mahdollinen käsitteellinen väline tilanteen tarkastelemiselle on *muuttujan roolin* käsite (Sajaniemi & Kuittinen, 2005). Muuttujien rooleja käyttäen mahdollinen selitys asetelmalle olisi, että viikolla 2 haastateltava B tunnisti ohjelmasta tutun roolin ”laskuri” (tai ”askeltaja”, *stepper*) ja tämän roolitietämyksen avulla onnistui nostamaan simulointinsa abstraktiotasoa. Vastaavasti viikolla 3 edellä mainitun kvalitatiivisen päätelmän tekoon olisi tarvittu ymmärrystä muuttujan roolista ”tilapäissäilö”

(*temporary*), mutta jos B ei tuntenut tällaista muuttujan roolia, päätelmän teko ei onnistunut.

Saman koodiesimerkin simuloinnin yhteydessä, kun kahdesta sisäkkäisestä silmukasta ulomman silmukan toimintaa oli simuloitu yksi iteraatio, B esitti pyydettäessä ”käsitteellisellä tasolla” päätelmän metodin toiminnasta. Päätelmä oli oikeansuuntainen, mutta ei oikea. B:n kuvaus vastasi eri järjestämisalgoritmin toimintaa kuin minkä koodi toteutti. Vaikka siis operationaalinen ymmärrys oli jo saavutettu sillä tasolla, että koodia pystyttiin simuloimaan, tästä huolimatta koodin funktion päättelyssä tapahtui virhe.

Luvussa 4.3.3 raportoitu C:n viikolla 2 käyttämä yhden arvon simulointistrategia on selkeä esimerkki aloittelijan käyttämästä simulointistrategiasta, joka ei sisälly esimerkiksi Détiennen ja Solowayn tunnistamiin strategioihin, ja jota ei muutenkaan ole aiemmin käsitelty kirjallisuudessa. Vaikka tässä tutkimuksessa saatiin evidenssiä yhden arvon simulointistrategian käytöstä vain yhdessä tilanteessa, strategiaa voinee pitää uskottavana ja melko todennäköisesti myös yleisenä sen äärimmäisen yksinkertaisuuden takia. Kognitiivisesti strategia edellyttää vain yhden arvon ylläpitoa kerrallaan muistissa, ja strategia ei edellytä juuri lainkaan ohjelmointitietoa (kuten skeemoja), kuten monet muut strategiat. Yhden arvon simulointistrategia on luonnollisesti myös virhealtis (kuten C:n käyttäytymisestä havaittiin), ja se tuskin soveltuu kovin monimutkaisten ohjelmien käsittelyyn. Voitanee kuitenkin olettaa, että opintojen alkuvaiheessa tätä kyseistä strategiaa käyttää usea opiskelija. Tätä strategiaa voinee pitää heuristiikkana, ”kokeillaan, onnistuuko simulointi näin helposti”. Jos osoittautuu, että simulointia vaativa ongelma ei ratkea tällaisella yksinkertaisella strategialla, henkilö voi siirtyä käyttämään jotakin huolellisempaa strategiaa. Tällaiseen kokeiluun liittyy kuitenkin sellainen riski, että tämä yksinkertainen strategia tuottaa virheen, mutta virhettä ei huomata. Tällaisen strategian olemassaolon tuntemisesta saattaa ohjelmoinnin opettajille olla hyötyä heidän selvitellessään opiskelijoiden kohtaamia ongelmatilanteita. Strategian käytöstä pois opastaminen voi tarvittaessa tapahtua yksinkertaisesti niin, että pelkästään arvojen tarkastelun lisäksi kiinnitetään huomio siihen, mihin tallennuspaikkaan eli muuttujaan mikäkin arvo on tallennettu.

Viikon 6 haastattelussa C käytti toisenlaista heuristista strategiaa, jossa hän simuloi koodia valikoivasti. Simuloinnin alkuvaiheessa hän ei tarkastellut while-lauseen ehto-osaa lainkaan, mutta käsitteli while-lauseen sisällä olleet lauseet. Myöhemmin simuloinnissaan, kun

haastattelija oli maininnut C:n tekemästä virheestä, C muodosti ohjelman toimintaan liittyvän melko mielikuvituksellisen hypoteesin, jonka avulla hän yritti selittää ohjelman toimintaa. Haastattelijan kysyessä perusteluja hypoteesille, C ei pystynyt esittämään perusteluja. C siis turvautui strategiaan, jossa hän muodosti ohjelman toimintaa funktionaalisella tasolla selittävän hypoteesin ja yritti sen avulla selittää ohjelman toimintaa top-down-tyyliin. Vaihtoehtona tälle olisi B:n käyttämä bottom-up-tyylinen strategia, jossa simuloidaan koodin suoritusta lause lauseelta. B ei oma-aloitteisesti esittänyt funktionaalisia päätelmiä, mutta samalla viikolla simuloinnin jälkeen kysyttäessä pystyi esittämään sellaisen.

Syy C:n käyttämään strategiaan jäi spekuloinnin varaan. C:n lausumista asioista voitiin päätellä, että ainakin syntaktisesti hän kyllä ymmärsi esimerkiksi mainitun while-lauseen ja olisi siten luultavasti pystynyt konkreettisesti simuloimaan ohjelman toimintaa. Aivan simuloinnin loppuvaiheessa, kun haastattelija oli maininnut, että silmukka käydään useaan kertaan läpi, C oma-aloitteisesti esitti selkeän ja oikean päätelmän silmukan funktiosta. Yksi selitys heuristisen strategian käytölle saattaisi olla pyrkimys suoriutua tehtävästä nopeasti ja tehokkaasti, ilman vaivalloista simulointia. C:n tekemä hypoteesin muodostaminen ja sen arviointi eivät kuitenkaan olleet tehokkaita tapoja selvittää ohjelman toimintaa, vaan aiheuttivat paljon prosessointia. On mahdollista, että päätökseen strategian valinnasta vaikuttaa jonkinlainen heuristinen arvio strategioiden käytön vaatiman prosessoinnin määrästä. Jos esimerkiksi olisi odotettavissa, että toistolauseen simuloimiseksi jouduttaisiin simuloimaan lukuisia iteraatioita, henkilöllä on kiusaus yrittää päätellä toistolauseen vaikutukset ilman simulointia. Jos taas etukäteen pystyttäisiin päättelemään, että toistolauseesta joudutaan simuloimaan vain pieni määrä iteraatioita, kiusaus muun strategian käyttöön olisi vastaavasti pienempi. Tämä pohdinta jää spekulaatioksi, koska strategioiden käytön laukaisevia ehtoja (varsinkaan odotettuun työmäärään perustuvia ehtoja) ei ohjelmoinnin psykologiassa ole kirjoittajan tietojen mukaan tutkittu. C:n simuloinnista viikolta 6 voitiin kuitenkin todeta, että C muutenkin esitti koodista laadullisia päätelmiä, esimerkiksi ”...jos tää ohjelma alkaa tulostelemaan, ku tää ohjelma ajetaan, ni se on luonu ensin nää uudet, uudet Lista-oliot, sit se printtaa että haa...” Tällaisten päätelmien esittäminen osoittaa pyrkimystä hahmottaa koodia korkeammalla tasolla kuin vain yksittäisiä lauseita simuloitaessa.

5.2.6. Opiskelijoiden funktionaalinen tieto

Edellä on todettu, että onnistuneeseen simulointiin tarvitaan täsmälliset tiedot ohjelman lauseiden toiminnasta. Pelkästään tiedot lauseiden käyttötarkoituksesta tai mekaaniset taidot simuloinnissa käytettävästä muuttujien arvojen kirjaamisesta eivät riitä. Luvussa 4.3.5 nähtiin, että C:n simulointia vaikeutti se, että hän oli sekoittanut for-lauseen toimintaan ja sen yleiseen käyttötarkoitukseen liittyvät tiedot keskenään. Samassa luvussa nähtiin myös, miten viikolla 1 A:n vastaukset sisälsivät vain funktionaalisia seikkoja niistä käsitteistä, joita haastattelija pyysi häntä määrittelemään. A:n vastaustyyli saattaa kuvastaa hänen ennakko-oletustaan siitä, millaisia vastauksia haastattelija odottaa, eikä pelkästään sitä, millaisia tietoja hän on omaksunut. Kuitenkin jos vastaukset johtuivat tällaisesta ennakko-oletuksesta, oletuksen on kuvastettava sitä, millaista tietoa A ajatteli pidettävän olennaisena. Tietyn tyylisen tiedon korostaminen saattaa ilmentää myös A:n aiemmin omaksumaa opiskelutekniikkaa. Näistä mahdollisista syistä huolimatta voidaan ajatella, että joka tapauksessa opiskelijoiden käsityksiä olisi selventänyt, jos kurssilla olisi selvästi tuotu esiin se seikka, että käsiteltävistä käsitteistä esitetään sekä toimintaan liittyviä tietoja että käyttötarkoitukseen liittyviä tietoja, ja että toimintaan liittyviä tietoja tarvitaan ohjelmien toiminnan selvittämiseen ja käyttötarkoitukseen liittyviä tietoja tarvitaan edellisten lisäksi apuna ohjelmia suunniteltaessa.

5.2.7. Opiskelijoiden terminologiset ongelmat

Molemmissa viikon 3 haastatteluissa tuli esiin, että kurssilla käytetty terminologia aiheuttaa ongelmia opiskelijoille. Kielen tasoa syvemmällä tasolla tarkasteltuna vaikuttaa siltä, että ylipäätään käsitteiden hahmottaminen tuottaa opiskelijoille ongelmia. Viikolla 3 B:llä ongelmat vaikuttivat olevan lähinnä pelkästään termien tasolla, mutta A:n käsityksissä oli havaittavissa perustavanlaatuisia käsitteellisiä sekaannuksia, esimerkkinä luokan käsitteen ja pääohjelman käsitteen sekoittuminen toisiinsa. Aiemmin viikolla 1 B:n havaittiin ymmärtäneen väärin **syötteen** käsitteen.

Eräs mahdollinen hypoteesi ongelman syystä liittyy siihen, että ohjelmoinnista puhuttaessa käytössä voi olla useita eri konteksteja, esimerkiksi konkreettisten operaatioiden konteksti ja toisaalta ohjelmien suunnittelun konteksti. Konkreettisessa kontekstissa puhe koskee

ohjelmakoodia sekä selkeästi määriteltyjä operaatioita tai dataa ja puheessa esiintyvät viittaukset viittaavat ohjelmakoodista selkeästi tunnistettaviin osiin. Konkreettisessa kontekstissa relevantit operaatiot ovat ohjelman suoritukseen liittyviä operaatioita, kuten metodikutsuja, olioiden luontia tai arvojen sijoittamista. Pystyäkseen ymmärtämään konkreettisen tason puhetta tulee tuntea näiden operaatioiden toiminnan mekanismit eli käsitteellistä tai teoreettista tietoa ohjelmien toiminnasta. Suunnittelukontekstissa taas ohjelmaa tarkastellaan abstraktimpien yksiköiden tasolla ja puheessa käytettävät viittaukset saattavat viitata myös sovellusalueen käsitteisiin (sovellusaluetta koskevaa tietoa käsiteltiin luvussa 1.3.3) sen sijaan, että viittaisivat ohjelman käsitteisiin. Relevantit operaatiot saattavat sisältää sovellusalueen operaatioita tai ohjelman toimintaan liittyviä operaatioita ilmaistuna sellaisella abstraktiotasolla, joka vastaa sovellusalueen operaatioita.

Seuraavassa havainnollistetaan esitettyä hypoteesia A:n haastatteluista tehdyillä havainnoilla. Viikon 3 haastattelussa haastateltava A ei osannut vastata kysymykseen, ovatko aksessorit metodeja. Metodi on konkreettinen koodista löytyvä osa, kun taas aksessorista puhutaan yhteydessä, jossa käsitellään abstraktiksi tietotyypiksi suunnitellun olion tietosisältöä. Käsitteet eivät ole siinä mielessä synonyymejä, että kaikki metodit eivät ole aksessoreja. Olisi luultavasti mahdollista luoda esimerkkitilanne, jossa olisi tulkinnanvaraista, voidaanko jotakin metodia kutsua aksessoriksi vai ei. A luultavasti oli kuullut termejä käytettävän eri asiayhteyksissä, eikä ollut pystynyt muodostamaan selkeää yhteyttä niiden välille.

5.2.8. Opetuskieli ja opetusjärjestys

Tietojenkäsittelytieteen opettajien parissa käydään paljon keskustelua siitä, missä järjestyksessä aloittaville tietojenkäsittelytieteen opiskelijoille tulisi asioita opettaa ja mikä olisi sopiva ensimmäinen ohjelmointikieli ohjelmoinnin opetuksen aloittamiseen (esimerkiksi Bruce, 2004). Näissä keskusteluissa valintojen perusteena ovat kuitenkin useimmiten yksittäisten opettajien taitoihin, mielipiteisiin ja mieltymyksiin liittyvät perusteet, opetustarjonnan suunnitteluun liittyvät perusteet sekä työelämän tarpeisiin liittyvät perusteet. Näissä keskusteluissa ei käytännössä lainkaan käytetä psykologisia perusteita väitteille, joten näiden keskustelujen tarkastelu kognitiotieteellisessä tutkielmassa ei olisi mielekästä. Mielenkiintoinen kysymys on, olisiko psykologisen tiedon perusteella mahdollista ottaa kantaa

mainitussa keskustelussa pohdittaviin kysymyksiin. Tämän tutkimuksen perusteella näkemys on, että psykologisin tutkimusmenetelmin ja psykologisen tiedon perusteella voidaan ottaa kantaa esimerkiksi siihen, mitä ongelmia tietyssä oppisisällön esitystavassa tai ohjelmointikielen rakenteessa on ja miten ongelmia voisi korjata. Tällöin puhutaan hyvin yksityiskohtaisista asioista yksittäisessä tietyssä oppimateriaalissa. Mainitut keskustelut eivät kuitenkaan liiku minkään tiettyjen esitystapojen tasolla, vaan hyvin laajojen lähestymistapojen välillä siten, että käsiteltävien vaihtoehtojen sisälle mahtuu laaja joukko hyvin erilaisia tapoja esittää oppisisältöjä. Tämän vuoksi tässä tutkielmassa ei oteta kantaa siihen, mitä ohjelmointikieltä pitäisi käyttää ensimmäisenä opetuskielenä tai pitäisikö olio-ohjelmoinnin käsitteet opettaa kurssin alku- vai loppupuolella.

Esimerkki yksityiskohtaisesta kielen rakenteeseen liittyvästä havainnosta on luvussa 4.1.1 mainittu ongelma siitä, miten for-lause esitetään Java-kielessä. For-lauseen määrittelyssä on kolme osaa, jotka erotetaan toisistaan puolipisteellä. Kaikkialla muualla Java-kielessä puolipisteellä erotetaan toisistaan kokonaiset lauseet, jotka suoritetaan peräkkäin toistensa jälkeen. For-lauseen tapauksessa puolipisteellä toisistaan erotettujen lausekkeiden suoritusjärjestys on for-lauseen logiikan mukainen ja poikkeaa kaikkialla muualla toteutuvasta peräkkäisestä järjestyksestä. Puolipisteen käyttö erotinmerkkinä selvästi johti joitakin haastateltavia harhaan for-lausetta käsiteltäessä. Kieltä suunniteltaessa ongelma olisi voitu korjata määrittelemällä for-lauseelle sellainen syntaksi, että samaa symbolia ei käytetä eri paikoissa eri merkityksissä. Nyt Java-kieltä opetettaessa ongelmaa voidaan yrittää korjata kiinnittämällä erityistä huomiota tämän hämmentävän piirteen selittämiseen. Kuten esimerkistä voi huomata, tämän yksityiskohtaisen havainnon perusteella tuskin on mielekästä ottaa kantaa siihen, kannattaako Java-kieltä käyttää ensimmäisenä opetuskielenä tai missä järjestyksessä kielen rakenteita kannattaa kurssilla opettaa.

5.3. Tulevia tutkimussuuntia

Ohjelmoinnin psykologian tutkimukselle voidaan määrittää tavoitteita esimerkiksi tietojenkäsittelytieteen pedagogiikan tai kognitiotieteen kannalta. Tässä tutkielmassa pyrittiin täyttämään molempiin aloihin liittyviä tavoitteita. Tutkielman aiheen laajuuden ja eksploratiivisen luonteen vuoksi saatuja tuloksia on lukumäärällisesti paljon ja ne liittyvät useisiin eri aiheisiin. Kutakin seuraavassa mainittua asiaa voidaan käyttää jatkotutkimuksen pohjana

ottamalla asia tarkemman tutkimuksen kohteeksi tai käyttämättä tässä tehtyjä havaintoja hyödyksi muodostettaessa uusia hypoteeseja.

Tietojenkäsittelytieteen pedagogiikan kannalta ohjelmoinnin psykologia voi ottaa kantaa siihen, millaiset tiedot ja millainen tietojen järjestyminen mielessä auttavat oppilaita suoriutumaan niistä tehtävistä, joista heidän oppimistavoitteiden mukaan halutaan suoriutuvan. Vastaavasti voidaan ottaa kantaa siihen, miten oppisisältöjä tulisi esittää, jotta oppilaiden mieleen muodostuisi halutunlaisia käsityksiä. Tässä tutkielmassa tuotiin esiin opiskelijoiden virhekäsityksiä sekä pohdittiin, millaisilla keinoilla virhekäsitysten muodostumista voitaisiin välttää. Tällaisia keinoja ovat esimerkiksi havaintoesitysten ja käsitteellisten mallien käyttäminen sekä esitettävän tiedon järjestäminen niin, että operationaalinen ja funktionaalinen tieto pidetään selvästi erillään toisistaan. Näistä operationaalisen ja funktionaalisen tiedon erillään pitämistä ei ole aiemmin käsitelty ohjelmoinnin psykologiassa.

Kognitiotieteen kannalta tutkimuksen tavoite on lisätä ymmärrystä siitä, millaista tietoa ohjelmoijilla on mielessään, miten tämä tieto on järjestynyt, miten tiedot muuttuvat oppimisen myötä ja miten tietoja käytetään ajattelussa. Tämän tutkielman tavoite oli dokumentoida sellaisia opiskelijoiden virhekäsityksiä, joihin ei aiemmin ole tutkimuksessa kiinnitetty huomiota. Ohjelmointikielen käsitteisiin liittyviä tuloksia olivat tyypin ja pääohjelman käsitteisiin liittyvien ongelmien, viitesemantiikan ymmärtämiseen liittyvien ongelmien sekä terminologisten ongelmien dokumentointi. Tietojen käyttöön liittyviä tuloksia olivat yhden esimerkin perusteella tehdyn yliyleistämisen tarkasteleminen, minimalistisen ja luonteeltaa heuristisen ”yhden arvon simulointistrategian” dokumentoiminen, simuloinnin abstraktiotason nostamisen edellytysten tarkasteleminen sekä rakenteen ja toiminnan sekoittamisen tarkasteleminen. Näistä erityisesti "yhden arvon simulointistrategia" on asia, jota ei aiemmin ole käsitelty kirjallisuudessa.

Tehtäessä tutkimusta ihmisten väärinkäsityksistä voitaisiin ajatella, että mahdollisia tapoja ymmärtää jokin asia väärin olisi ääretön määrä. Kuitenkin kun yksittäisten havaintojen abstraktiotasoa nostetaan triviaalien yksittäisten havaintojen yläpuolelle, loogisesti toisiinsa liittyviksi havaintokokonaisuuksiksi, asia muuttuu. Ohjelmointikielen rakenteita on äärellinen määrä ja peruskurssilla niistä käsitellään vain osa. Samoin tutkijalla on käytössään äärellinen määrä kognitiivisen toiminnan hahmottamiseen liittyviä käsitteitä, joista tässä olivat käytössä esimerkiksi mentaalinen malli, skeema, mentaalinen simulointi, stra-

tegia ja yleistäminen. Tästä voidaan päätellä, että mahdollisten tutkittavien havaintokokonaisuuksien määrä on rajallinen. Tarkoitus ei ole väittää, että tässä tutkimuksessa löydettiin kattava joukko mahdollisista havaintokokonaisuuksista, vaan että käytetty tutkimusasetelma on mielekäs. Jos käytettyä tutkimusasetelmaa käytettäisiin useita kertoja, mahdollisesti eri tutkijoiden toimesta, eri kursseilla ja eri maissa, useiden tutkimusten tulosten olisi mahdollista konvergoitua suppeaksi joukoksi tarkasteltavia havaintokokonaisuuksia. Jos useiden tutkimusten tulokset saadaan konvergoitumaan eivätkä ne jää toisistaan irrallisiksi, pystytään kartuttamaan koherenttia ymmärrystä tutkimuksen kohteesta. Mahdollisuus useiden tutkimusten tulosten konvergoitumisesta koherentiksi ymmärrykseksi tekee tutkimuksen teon mielekkääksi.

Lähteet

Adelson, B. (1981). Problem solving and the development of abstract categories in programming languages. *Memory & Cognition*, 9(4), 422-433.

Adelson, B. & Soloway, E. (1985). The role of domain experience in software design. *IEEE Transactions on Software Engineering*, 11(11), 1351-1360.

Anderson, J. R. (1982). Acquisition of cognitive skill. *Psychological Review*, 89(4), 369-406

Anderson, J. R., Greeno, J. G., Kline, P. J. & Neves, D. M. (1981). Acquisition of problem-solving skill. Teoksessa J. R. Anderson (toim.), *Cognitive Skills and Their Acquisition* (s. 191-230). New Jersey: Lawrence Erlbaum Associates.

Anfara, V. A., Brown, K. M. & Mangione, T. L. (2002). Qualitative analysis on stage: Making the research process more public. *Educational Researcher*, 31(7), 28-38.

Bayman, P. & Mayer, R. E. (1983). A diagnosis of beginning programmers' misconceptions of BASIC programming statements. *Communications of the ACM*, 26(9), 677-679.

Brooks, R. (1977). Towards a theory of the cognitive processes in computer programming. *International Journal of Man-Machine Studies*, 9(6), 737-751.

Brooks, R. (1983). Towards a theory of the comprehension of computer programs. *International Journal of Man-Machine Studies,* 18(6), 543-554.

Brooks, R. (1990). Categories of programming knowledge and their application. *International Journal of Man-Machine Studies,* 33(3), 241-246.

Bruce, C., Buckingham, L. & Hynd, J. M. (2004). Ways of experiencing the act of learning to program: A phenomenographic study of introductory programming students at university. *Journal of Information Technology Education,* 3, 143-160.

Bruce, K. B. (2004). Controversy on how to teach CS 1: A discussion on the SIGCSE-members mailing list. *ACM SIGCSE Bulletin,* 36(2), 29-34.

Catrambone, R. (1994). Improving examples to improve transfer to novel problems. *Memory & Cognition,* 22(5), 606-615.

Catrambone, R. (1996). Generalizing solution procedures learned from examples. *Journal of Experimental Psychology: Learning, Memory, and Cognition,* 22(4), 1020-1031.

Clement, J. (1983) A conceptual model discussed by Galileo and used intuitively by physics students. Teoksessa D. Gentner & A. L. Stevens (toim.), *Mental Models* (s. 325-340). New Jersey: Lawrence Erlbaum Associates.

Davies, S. P. (1993). Models and theories of programming strategy. *International Journal of Man-Machine Studies,* 39(2), 237-267.

Détienne, F. (2002). *Software design — Cognitive aspects.* Lontoo: Springer.

Détienne, F. & Soloway, E. (1990). An empirically-derived control structure for the process of program understanding. *International Journal of Man-Machine Studies,* 33(3), 323-342.

diSessa, A. A. (1983). Phenomenology and the evolution of intuition. Teoksessa D. Gentner & A. L. Stevens (toim.), *Mental Models* (s. 15-34). New Jersey: Lawrence Erlbaum Associates.

Eckerdal, A. & Beglund, A. (2005). What does it take to learn 'programming thinking'?. Teoksessa *Proceedings of the 2005 international workshop on Computing education research* (s. 135-142). ACM Press.

Eckerdal, A. & Thuné, M. (2005). Novice Java programmers' conceptions of ”object” and ”class”, and variation theory. Teoksessa *Proceedings of the 10th annual SIGCSE*

conference on Innovation and technology in computer science education (s. 89-93). ACM Press.

Ericsson, K. A. & Simon, H. A. (1984). *Protocol analysis: Verbal reports as data.* MIT Press.

Fitzgerald, S., Simon, B. & Thomas, L. (2005). Strategies that students use to trace code: An analysis based in grounded theory. Teoksessa *Proceedings of the 2005 international workshop on Computing education research* (s. 69-80). ACM Press.

Fleury, A. E. (1991). Parameter passing: The rules the students construct. *ACM SIGCSE Bulletin,* 23(1), 283-286.

Fleury, A. E. (2000). Programming in Java: Student-constructed rules. Teoksessa *Proceedings of the thirty-first SIGCSE technical symposium on Computer science education* (s. 197-201). ACM Press.

Fleury, A. E. (2001). Encapsualtion and reuse as viewed by java students. Teoksessa *Proceedings of the thirty-second SIGCSE technical symposium on Computer Science* (s. 189-293). ACM Press.

Gentner, D. & Stevens, A. L. (1983). *Mental models.* New Jersey: Lawrence Erlbaum Associates.

Greeno, J. G. (1983). Conceptual entities. Teoksessa D. Gentner & A. L. Stevens (toim.), *Mental Models* (s. 227-252). New Jersey: Lawrence Erlbaum Associates.

Gries, P. & Gries, D. (2002). Frames and folders: A teachable memory model for Java. *Journal of Computing in Small Colleges,* 17(6), 182-196.

Hirsjärvi, S. & Hurme, H. (2000). *Tutkimushaastattelu — Teemahaastattelun teoria ja käytäntö.* Helsinki: Yliopistopaino.

Holland, S., Griffiths, R. & Woodman, M. (1997). Avoiding object misconceptions. *ACM SIGCSE Bulletin,* 29(1), 131-134.

Hundhausen, C. D. & Brown, J. L. (2005). Personalizing and discussing algorithms within CS1 studio experiences: An observational study. Teoksessa *Proceedings of the 2005 international workshop on Computing education research* (s. 45-56). ACM Press.

Hundhausen, C. D., Douglas, S. A. & Stasko, J. T. (2002). A meta-study of algorithm visualization effectiveness. *Journal of Visual Languages and Computing,* 13(3), 259-290.

Johnson-Laird, P. N. (1983). *Mental models.* Cambridge: Harvard University Press.

Joni, S. A. & Soloway, E. (1986). But my program runs! Discourse rules for novice programmers. *Journal of Educational Computing Research,* 2(1), 95-125.

Kempton, W. (1986). Two theories used of home heat control. *Cognitive Science,* 10, 75-91.

Kintsch, W. (1988). The role of knowledge in discourse comprehension: A construction-integration model. *Psychological Review,* 95(2), 163-182.

de Kleer, J. & Brown, J. S. (1981). Mental models of physical mechanisms and their acquisition. Teoksessa J. R. Anderson (toim.), *Cognitive Skills and Their Acquisition* (s. 285-310). New Jersey: Lawrence Erlbaum Associates.

de Kleer, J. & Brown, J. S. (1983). Assumptions and ambiguities in mechanistic mental models. Teoksessa D. Gentner & A. L. Stevens (toim.), *Mental Models* (s. 155-190). New Jersey: Lawrence Erlbaum Associates.

Klein, G. (1998). *Sources of power: How people make decisions.* MIT Press.

Lahtinen, E., Ala-Mutka, K. & Järvinen, H. (2005). A study of the difficulties of novice programmers. Teoksessa *Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education* (s. 14-18). ACM Press.

Lattu, M., Meisalo, V. & Tarhio, J. (2001). On using a visualization tool as a demonstration aid. Teoksessa *Proceedings of the First Program Visualization Workshop* (s. 141-162). Porvoo, Suomi. 7-8.7.2000. Joensuun Yliopisto.

Levy, R. B., Ben-Ari, M. & Uronen, P. A. (2003). The Jeliot 2000 program animation system. *Computers & Education,* 40(1), 1-15.

Lister, R., Adams, E. S., Fitzgerald, S. F., Fone, W., Hamer, J., Lindholm, M., McCartney, R., Moström, J. E., Sanders, K., Seppälä, O., Simon, B. & Thomas, L. (2004). A multi-national study of reading and tracing skills in novice programmers. *ACM SIGCSE Bulletin,* 36(4), 119-150.

Markman, A. B. & Gentner, D. (2000). Thinking. *Annual Review of Psychology,* 52, 223-247.

Mayer, R. E. (1975). Different problem-solving competencies established in learning computer programming with and without meaningful models. *Journal of Educational Psychology*, 67(6), 725-734.

Mayer, R. E. (1979). A psychology of learning BASIC. *Communications of the ACM*, 22(11), 589-593.

Mayer, R. E. (1981). The psychology of how novices learn computer programming. *Computing Surveys*, 13(1), 121-141.

Mayer, R. E. (1987). Cognitive aspects of learning and using a programming language. Teoksessa J. M. Carroll (toim.) *Interfacing Thought: Cognitive Aspects of Human-Computer Interaction* (s. 61-79). MIT Press.

McCauley, R. A. & Manaris, B. (2002). Comprehensive report on the 2001 survey of departments offering CAC -accredited degree programs. Tekninen raportti 2002-9-1, Department of Computer Science, College of Charleston.

Mellin, I. (1996). *Johdatus tilastotieteeseen, 1. kirja.* Tilastotieteen laitos, Helsingin yliopisto.

Neves, D. M. & Anderson, J. R. (1981). Knowledge compilation: Mechanisms for the automatization of cognitive skills. Teoksessa J. R. Anderson (toim.), *Cognitive Skills and Their Acquisition* (s. 57-84). New Jersey: Lawrence Erlbaum Associates.

Norman, D. A. (1983). Some observations on mental models. Teoksessa D. Gentner & A. L. Stevens (toim.), *Mental Models* (s. 7-14). New Jersey: Lawrence Erlbaum Associates.

Pea, R. D., (1986). Language-independent conceptual ”bugs” in novice programming. *Journal of Educational Computing Research*, 2(1), 25-36.

Pennington, N. (1987). Comprehension strategies in programming. Teoksessa G. M. Olson, S. Sheppard & E. Soloway (toim.), *Empirical Studies of Programmers: Second Workshop* (s. 100-113). New Jersey: Ablex.

Perkins, D. N., Hancock, C., Hobbs, R., Martin, F. & Simmons, R. (1986). Conditions of learning in novice programmers. *Journal of Educational Computing Research*, 2(1), 37-55.

Perkins, D. N. & Martin, F. (1986). Fragile knowledge and neglected strategies in novice programmers. Teoksessa E. Soloway & S. Iyengar (toim.) *Empirical Studies of Programmers: First Workshop* (s. 213-229). New Jersey: Ablex.

Rist, R. S. (1986). Plans in programming: Definition, demonstration, and development. Teoksessa E. Soloway & S. Iyengar (toim.) *Empirical Studies of Programmers: First Workshop* (s. 28-47). New Jersey: Ablex.

Rist, R. S. (1989). Schema creation in programming. *Cognitive Science,* 13(3), 389-414.

Roman, G. & Cox, K. C. (1993). A taxonomy of program visualization systems. *IEEE Computer,* 26(12), 11-24.

Saariluoma, P. (1988). Ohjelmoinnin psykologia ongelmanratkaisun psykologian osa-alueena. *Psykologia,* 23(4), 262-272.

Saariluoma, P. (2001). Psychological problems in program visualization. Teoksessa *Proceedings of the First Program Visualization Workshop* (s. 13-27). Porvoo, Suomi. 7-8.7.2000. Joensuun Yliopisto.

Sajaniemi, J. (2002). An empirical analysis of roles of variables in novice-level procedural programs. Teoksessa *Proceedings of IEEE 2002 Symposia on Human Centric Computing Languages and Environments* (s. 37-39). IEEE Computer Society.

Sajaniemi, J. & Kuittinen, M. (2005). An experiment on using roles of variables in teaching introductory programming. *Computer Science Education,* 15(1), 59-82.

Scott, M. L. (2000). *Programming language pragmatics.* San Francisco: Morgan Kaufmann.

Soloway, E. (1986). Learning to program = learning to construct mechanisms and explanations. *Communications of the ACM,* 29(9), 850-858.

Soloway, E., Bonar, J. & Ehrlich, K. (1983). Cognitive strategies and looping constructs: An empirical study . *Communications of the ACM,* 26(11), 853-860.

Soloway, E. & Ehrlich, K. (1984). Empirical studies of programming knowledge. *IEEE Transactions on Software Engineering,* 10(5), 595-609.

Soloway, E., Ehrlich, K. & Black, J. B. (1983). Beyond numbers: Don't ask ”how many” ... ask ”why”. Teoksessa *Proceedings of the SIGCHI conference on Human Factors in Computing Systems* (s. 240-246). ACM Press.

Soloway, E., Ehrlich, K. & Bonar, J. (1982). Tapping into tacit programming knowledge. Teoksessa *Proceedings of the 1982 conference on Human factors in computing systems* (s. 52-57). ACM Press.

Sommerville, I. (2001). *Software engineering, Sixth Edition.* Pearson, Englanti.

Spohrer, J. C. & Soloway, E. (1986). Novice mistakes: Are the folk wisdoms correct?. *Communications of the ACM,* 29(7), 624-632.

Spohrer, J. C., Soloway, E. & Pope, E. (1985). A goal/plan analysis of buggy pascal programs. *Human-Computer Interaction,* 1, 163-207.

Stasko, J., Badre, A. & Lewis, C. (1993). Do algorithm animations assist learning? An empirical study and analysis. Teoksessa *Proceedings of the SIGCHI conference on Human factors in computing systems* (s. 61-66). ACM Press.

Sun Microsystems, Inc. (1999). Code conventions for the java programming language. http://java.sun.com/docs/codeconv/ [5.5.2006].

Thomas, L., Ratcliffe, M. & Thomasson, B. (2004). Scaffolding with object diagrams in first year programming classes: Some unexpected results. *ACM SIGCSE Bulletin,* 36(1), 250-254.

Van Dijk, T. A. & Kintsch, W. (1983). *Strategies of discourse comprehension.* Academic Press, New York.

Wikla, A. (2001). *Ohjelmoinnin perusteet Java-kielellä.* Otadata, Espoo.

Wikla, A. (2004). Ohjelmoinnin perusteita Java-kielellä. http://www.cs.helsinki.fi/u/wikla/JohdOhj/Sisalto/index.html [31.3.2006].

Liite A: Koodinäytteet

Koodinäyte A1: Luokka1

Tätä koodinäytettä käsiteltiin haastatteluviikolla 1.

```
public class Luokka1 {
	static final int SUURIN = 9;

	public static void main(String[] args) {
		int pieni = 1;
		int iso = 1;
		String merkki;

		System.out.println("1: " + pieni);
		for (int i = 2; i <= SUURIN; i++) {
			if (iso % 2 == 0)
				merkki = " *";
			else
				merkki = "";

			System.out.println(i + ": " + iso + merkki);

			iso = pieni + iso;
			pieni = iso - pieni;
		}
	}
}
```

Koodinäyte A2: Luokka2

Tätä koodinäytettä käsiteltiin haastatteluviikolla 2.

```
public class Luokka2 {

	public static void main(String[] args) {
		int luku = 64;
		int laskuri = 0;

		for (int l = 0; luku > 2; luku /= 2) {
			for (int k = 0; k < luku; k++) {
				laskuri++;
				System.out.println(laskuri);
			}
		}
	}
}
```

Koodinäyte A3: TekeeJotain

Tätä koodinäytettä käsiteltiin haastatteluviikolla 3.

```
public class TekeeJotain {

    public static int[] teeJotain(int[] luvut) {
        int mihinAsti = luvut.length - 2;
        int parhaillaan = 0;
        int apu;

        while (mihinAsti >= 0) {
            parhaillaan = 0;
            while (parhaillaan <= mihinAsti) {
                if (luvut[parhaillaan] > luvut[parhaillaan+1]) {
                    apu = luvut[parhaillaan];
                    luvut[parhaillaan] = luvut[parhaillaan+1];
                    luvut[parhaillaan+1] = apu;
                }
                parhaillaan = parhaillaan + 1;
            }
            mihinAsti = mihinAsti - 1;
        }

        return luvut;
    }

}
```

Koodinäyte A4: KasitteleTaulukkoa

Tätä koodinäytettä käsiteltiin haastatteluviikolla 3.

```
public class KasitteleTaulukkoa {

    public static void main(String[] args) {
        // luodaan uusi taulukko,
        // jonka sisällöksi sijoitetaan luetellut luvut
        int[] taulukko = new int[] { 7, 12, 2, 9, 3, 18, 0 };

        int[] toinenTaulukko = TekeeJotain.teeJotain(taulukko);

        for (int i=0; i<taulukko.length; i++) {
            System.out.println(taulukko[i]+"\t"+toinenTaulukko[i]);
        }
    }

}
```

Koodinäyte A5: Lista

Tätä koodinäytettä käsiteltiin haastatteluviikolla 6.

```
public class Lista {

    private String sisältö;
    private Lista seuraava;

    public Lista(String sisältö) {
        this.sisältö = sisältö;
        this.seuraava = null;
    }

    public Lista(String sisältö, Lista seuraava) {
        this.sisältö = sisältö;
        this.seuraava = seuraava;
    }

    public String toString() {
        return sisältö;
    }

    public static void tulostaLista(Lista eka) {
        Lista nykyinen = eka;
        while (nykyinen != null) {
            System.out.print(" "+nykyinen.sisältö);
            nykyinen = nykyinen.seuraava;
        }
        System.out.println();
    }

    public static void main(String[] args) {
        Lista hip = new Lista("hip");
        Lista hap = new Lista("hap", hip);
        Lista huu = new Lista("huu");
        Lista juu = new Lista("juu", huu);
        Lista jaa = new Lista("jaa", hap);
        Lista haa = new Lista("haa", hap);

        System.out.print("haa: ");
        tulostaLista(haa);
        System.out.print("jaa: ");
        tulostaLista(jaa);
        System.out.print("juu: ");
        tulostaLista(juu);
        System.out.print("huu: ");
        tulostaLista(huu);
        System.out.print("hap: ");
        tulostaLista(hap);
        System.out.print("hip: ");
        tulostaLista(hip);
    }

}
```

Liite B: Sanalliset tehtävät

Liite B2: Tehtävä 2

Tehtävä 2

Kirjoita ohjelma, joka ensin kysyy käyttäjältä, montako lukua käyttäjä haluaa syöttää.

Seuraavaksi ohjelma pyytää käyttäjää syöttämään käyttäjän antaman määrän lukuja. (Saat ohjelmoida lukujen syötön haluamallasi tavalla. Voit käyttää Lue-luokkaa.)

Seuraavaksi ohjelma tulostaa kaikki käyttäjän syöttämät luvut luettelona, jossa joka luvun väliin tulostetaan pilkku.

Lopuksi ohjelma tulostaa tiedon siitä, mikä oli suurin syötetty luku, ja mikä oli pienin syötetty luku (myös negatiiviset luvut sallitaan).

Vihje: ohjelman on luultavasti syytä tallentaa luvut taulukkoon.

Liite C: Haastattelurungot

Tässä liitteessä esitetään haastattelijan käyttämät haastattelurungot siinä muodossa kuin ne otettiin käyttöön haastatteluviikon alussa. Mahdolliset viikon aikana tehdyt täydennykset eivät näy tässä liitteessä. Mahdolliset tekstin lomassa olevat huomautukset ovat osa haastattelurunkoja.

Liite C1: Viikon 1 haastattelurunko

Taustatiedot
- ikä
- suoritetut tutkinnot (matematiikka, kielitiede tms.?)
- aiempi tietokoneen käyttö
- aiempi ohjelmointiin liittyvä kokemus

- Motivointi - halutaan juuri kurssilaisten käsityksiä, joten haastateltava varmasti on oikea ja hyödyllinen henkilö tähän tarkoitukseen
- Halutaan asioille syitä ja perusteluja, ajatuskulkuja. On aikaa miettiä ja vastata rauhassa. Saa puhua pitkään.
- Jos on opetuksessa kuullut tai itse keksinut jotain vertauskuvia tms., niin niistä saa kertoa.
- Vaitiolo - ei saa puhua haastatteluista samalla viikolla, etteivät myöhemmät haastattelut mene pilalle.

Mitä olet oppinut?
- käsitteiden tarkennuksia

Määriteltävät käsitteet (missä, miksi käytetään?)
- ohjelma
- algoritmi
- muuttuja
- syöte
- tuloste
- sijoituslause
- vertailuoperaatio

Koodinäyte:
Luokka1.java
- nimeä mahdollisimman monta asiaa
- kerro, miten ohjelman suoritus etenee ja mitä ohjelma tulostaa
- miten voisit saada tietoa ohjelman suorituksesta?

HUOM: koodinäytteessä käytetään for-rakennetta, joka on jo ollut maanantain luennolla. Maanantain luennolla EI vielä ole käsitelty metodeja, joten koodinäytteessä ei niitä voi käyttää.

Liite C2: Viikon 2 haastattelurunko

Haastattelu 2

Edellisen haastattelun jälkeen opitut asiat?

Käsitteiden määrittelyä
- olio
- luokka
- metodi
- lause
- lauseke
- lohko
- algoritmin tila

Mitä tapahtuu silloin, kun ohjelmassa esitellään uusi muuttuja?

For-lauseen rakenne ja toiminta

Tehtävät:

Aluksi ilman apua!

Tehtävä 1
- ratkaise
- ajattele ääneen

Luokka2.java
- miten suoritus etenee? (ei tarvitse edetä lause lauseelta, saa myös päätellä)
- mitä tulostaa?

Liite C3: Viikon 3 haastattelurunko

Haastattelu 3

Eteneminen viime haastattelun jälkeen
- tehtävät
- termit

Määriteltävää:
- kapselointi
- olio
- aksessori
- oliotyyppinen muuttuja (viite olioon)
- arvon palauttaminen
- algoritmin tila

Mainitse muutama esimerkki lauseesta, joka luo uuden olion?

Miten pystyt ohjelmakoodia katsomalla selvittämään, mitä ohjelma tekee?

Jos ohjelmassasi on virhe, mitä tapoja voit käyttää virheen löytämiseksi?

Tehtävä 1:
Näytetään TekeeJotain-luokan koodi, ja kysytään mitä teeJotain-metodi tekee.
Seuraavaksi näytetään KasitteleTaulukkoa-luokan koodi, ja kysytään mitä ohjelma tulostaa.
(Ei sisäkkäisiä for-silmukoita, teeJotain on toteutettu while-silmukoilla. teeJotain-metodissa kaikki muuttujat alustetaan aluksi.)

Tehtävä 2: (erillisellä paperilla)

Liite C4: Viikon 4 haastattelurunko

Haastattelu 4

Eteneminen viime haastattelun jälkeen
- paikallaolo, tehtävät
- tentti
- termit, metaforat (metafora olioviitteistä?)
- käytetty aika? apu?
- aiempien opiskelujen vaikutus?

Määriteltävää:
- luokka
- konstruktori
- olion tila
- taulukon indeksi
(- arvon palauttaminen...)
- näkyvyys

Millainen on kaksiulotteisen taulukon rakenne? (mitä tapahtuu viitattaessa)

Virheen etsiminen ohjelmasta?

Tehtävät
1. PikkuOlio.java + SekavaLuokka.java
2. MetodiKutsu.java
3. ed. kerran Tehtävä 2 (paitsi D)

Arvio suoriutumisesta haastattelutilanteessa verrattuna muuhun työskentelyyn?

Liite C5: Viikon 5 haastattelurunko

Haastattelu 5

Eteneminen viime haastattelun jälkeen
- paikallaolo, tehtävät
- termit, metaforat
- käytetty aika? apu?
- aiempien opiskelujen vaikutus? (erityisesti haastateltava D)

Käsitteitä
- luokka, olio
- kapselointi (ja sen merkitys)
- oliomuuttuja, luokkamuuttuja
- metodi (kaikki mahdollinen toiminnasta)
- periytyminen (laajasti eri aspekteja)

Tehtävät
- Tehtävä 3
- luokka MetodiKutsu (ei D)
- Tehtävä 4 (D)

Liite C6: Viikon 6 haastattelurunko

Haastattelu 6

Eteneminen viime haastattelun jälkeen
- paikallaolo, tehtävät
- käytetty aika
- termit
- tentti
- lunttilappu

Käsitteitä
- luokka, olio
- rajapinta(-luokka), abstrakti luokka
- toteuttaminen (termi)
- poikkeukset (merkitys ja käyttö, aiheuttaminen ja sieppaaminen)
- try-catch-finally -rakenne (toiminta)

Mitä asioita tarkistetaan ohjelman kääntämisen aikana ja mitä tarkistetaan ohjelman suorituksen aikana? (sijoitus muuttujaan, toistolauseen ehdon tyyppi, muuttujan näkyvyys, taulukon indeksointi, viitteen tyyppimuunnos, metodikutsun parametrien tyyppi, metodikutsun kohde, arvon palautus metodista)

API-kuvausten lukeminen? (Lehmä, Eläin, mitä tietoja löytyy?)

(- lauseiden jäsentämiseen ja lauseensisäiseen suoritusjärjestykseen liittyvät asiat)
(- ohjelman suorituksen etenemiseen liittyvät asiat)
(- lohkorakenteeseen liittyvät asiat)
(edellisten muuttuminen kurssin aikana?)

Tehtävät:
- Lista.java (simulointi)
- Tehtävä 5 (korostaen, että ei tehdä mitään syöttöä tai tulostusta)

Tietojen käyttö
- ikä, sukupuoli, koulutustausta
- asiayhteys: lainaukset haastatteluista, tulkinnat ja päätelmät haastatteluista

Liite D: Esimerkki haastatteluprotokollasta

Tämä liite sisältää protokollan yhdestä haastattelusta kokonaisuudessaan. Kyseessä on haastateltavan B haastattelu viikolta 3.

H (2): Okei. Totanoin. Nyt on torstai ja sulloli viimeks keskiviikkona oli ennen laskareita haastattelu, eiks niin?
B (4): Joo.
H (4): Ja siinon nyt ollut välissä sit perjantain luento, maanantain laskarit, tiistain luento, keskiviikon laskarit.
B (6): Joo.
H (6): Joo. Eli kaks luentoo ja kahdet laskarit. Ooksä ollu näis kaikis paikalla?
B (7): En, yhet laskarit jäi väliin, maanantaina.
H (8): Joo. Noil luennoilla sä oot ollu?
B (8): Luennoilla oon... siis, perj..., eiku hetkinen, enkä oo, ku perjantain jäi myös väliin. Perjantain luennolla ja maanantain laskareissa en ollu.
H (10): Joo. Tiistain luennolla ja keskiviikon laskareissa...?
B (11): ...laskareissa, olin, joo.
H (11): Miten noi laskaritehtävät, teiksä molempien kertojen tehtävät?
B (11): Kyl mä oon ne kaikki tehny.
H (12): Ja onnistu tekeminen?
B (12): Joo, avustuksella.
H (13): Joo. Tota. Kuin paljon suunnilleen sä oot käyttäny aikaa per tollanen laskuharjoituskerta?
B (14): Laskuharjoituskerta?
H (15): Niin, tai, tai per päivä, tai miten sä nyt opiskelet?
B (15): No, *huokaa*, varmaan jos ajattelee niinku yhen laskuharjoituskerran tehtäviä ni varmaan kaheksasta kymmeneen tuntia yhteensä, ihan helposti. Enemmänki.
H (18): Joo. Joo. Sisält... Onks se pelkkää tehtävien tekoo vai sisältyyks siihen myös sen luentomateriaalin lukemista, tai...?
B (20): No siis sen verran miten tehtävien teon yhteydessä täytyy sitä lukee.
H (21): Joo. Joo. Ooksä sen lisäks muuten sit lukenu sitä materiaalia, erikseen?
B (22): No, en, en hirveesti. Sen verran nyt ku mä sen perjantain, viime perjantain luennon missasin, niin tota, sitä asiaa yrittäny ite kattoo.
H (23): Joo. Joo. Tota. Miltä noi tehtävät on tuntunu? Onks ne ollu...?
B (25): No, vaikeilta.
H (25): Oot selvinny kuitenki?
B (26): Joo.
H (26): Onks sul ollu tunne et sä oot ymmärtäny ne sitten ku sä oot saanu ratkaistua?
B (27): No joo, useimmissa tapauksissa kyllä.
H (27): Joo. Onks joku erityinen juttu jääny epäselväks?
B (28): Oo, no ei oikeestaan. Enemmän ne on ehkä semmosia, jotain tehtävien sanamuotoja, joita ei välttämättä oo oikein ymmärtäny, jossain tilanteessa.
H (30): Oisko sul esimerkkiä, et millasia sanamuotoja?
B (30): Aa, no siis, viime laskareissa oli tehtävä jossa piti muuttaa kokonais, tai siis tota, miten se nyt meni, niinku string-jono numeroita kokonaisluvuks, ja siinä pähkäilin pitkään, kun siinä tehtävän selostuksessa oli hirveesti, ku siinon paljon niinku sa..., toisiaan lähellä olevia käsitteitä, ja siinoli hirveesti sit niitä, niin semmosia sanoja, jotka niinku viittas eri käsitteisiin, mut lähes samoilla sanoilla, et oli numeromerkki ja numeromerkkiarvo ja numero ja luku.
B (38): Tämmösiä niinku siinä tehtävänannossa peräkkäin, niin meni aika pitkään aina välillä, siihen että mitäs, niinku mihinkä tässä nyt viitataan ku sanotaan luku, ja mihinkä tässä nyt viitataan ku sanotaan numeromerkki.
H (40): Joo.

B (40): Et sen tyyppistä, hahmottamista.
H (41): Joo. Onks sul muita tällasii sanoja, mitkä olis aiheuttanu pohdintaa?
B (43): Mmm, no siis ihan yleisesti nää, niinkun terminologia mitä käytetään näistä niinkun ohjelmoinnin, tai siis, tän kurssin puitteissa, mitä on oppinu näistä ohjelmointitermeistä, mitä nyt on, metodit ja aksessorit ja kentät ja tyypit ja muuttujat, tämmöset menee niinkun, kun niitä tulee niin paljon, ja suurin osa on täysin uutta terminologiaa, tai sitten tuttu sana, joka viittaa täysin niinkun uuteen asiaan, tai täysin toiseen asiaan ku mihin sen on tottunu viittaavan.
B (51): Niin sit ne vie aikaa kun tehtävänannossa, sit niinku laskaritehtävässä, lukee, ja kun niinkun käsketään käsittelemään jotain, jotain asiaa tietyllä tavalla, ni sen kelaamiseen, että mitäs tää nyt tarkottikaan.
B (54): Ja sitte vaikka puhutaan kentistä tai, tai vastaavista, niin sitten, se että yleensä täytyy aina konsultoida sitä materiaalia, että mitä se tämä nyt tarkottikaan tämä kenttä tässä yhteydessä, esimerkiks.
H (57): Joo. Onks luennoilla niin, luentojen seuraaminen onnistuu kuitenki hyvin, et...?
B (58): Joo, se on, se on, siellä kuitenki sitä itse terminologiaa, se käydään kyllä läpi, mutta se mennään niinkun aika nopeesti ohi, et sitte itse asia selitetään yleensä vähän toisilla sanoilla, koska niistä tulee niin paljon konkreettisia esimerkkejä.
H (62): Joo.
B (62): Niin sit ne ei ehkä ne termit tartu ihan samalla tavalla.
H (63): Joo. Nii eli luennoilla niin, näit termei ei käytetä niin paljon ku sitte...?
B (64): No siis kyllä niitä käytetään, mutta jotenkin, aa... Tai siis joo, siis ehkä niin, et, et esimerkiks ku luennoitsijalla on, öö, kun luentomateriaalissa luennoitsija on kehittäny tämmösiä ainakin omasta mielestään niinkun hyviä konkreettisia vertauksia, niin, niin sitten esimerkiks hän ei puhu esimerkiks metodeista välttämättä, vaan puhuu koko ajan pääohjelman pikku apulaisista, tai niinkun koneen kahvoista ja namiskuukkeleista, tai mitä siellä nyt onkaan näitä termejä.
H (71): Joo. Joo.
B (71): Et niinkun, et se on ehkä sit semmosta, tai siis luennolla tulee sit enemmän niinku asioista käytettyy niinku näitä, tämmösiä konkretiaversioita.
H (73): Joo. Aiheuttaaks se sitä sitte että, et nää tällaset abstraktimmat termit, niin ne jää vähän epäselväks?
B (75): Joo. Tai siis, siinon se, että ainaki mulla se fiilis on semmonen et ne, mä en hirveesti pidä siitä liiallisesta konkretisoimisesta, koska se jotenkin, niinkun, aa, siinä helposti mulla ainakin käy niin, että ne mielikuvat jää sille konkretian tasolle, kun se on hirveen konkreettista se, ne esimerkit ja ne vertaukset, niin sit, siit on vaikee päästä irti jotenki siitä konkretiasta tajuumaan se abstraktin tason siitä.
B (81): Et mulle puhutaan koko ajan oliosta koneena, esimerkiks, niin, niin mulle muodostuu niinku mielikuva koneesta, eikä välttämättä mielikuvaa siitä abstraktista asiasta, mikä on olio, ja kaikki ne muut mitä siihen kuuluu.
H (84): Joo.
B (84): Et se kyl sotkee, ainakin mua.
H (85): Joo. Joo. Tota, oisko sul vielä mielessä jotain esimerkkejä termeistä tai puhetavoista, jotka häiritsee...?
B (87): Totatota... No, tämä, kone-vertaus, esimerkiksi, mistä mä nyt äskenki puhuin, että, että luokka on koneen piirustukset ja olio, olio on sitten se kone itse, ja metodit on koneen kahvoja ja namiskuukkeleita, ja, ja, ja näin edelleen ja näin edelleen, ni se on, se on yks semmonen mikä mua sotkee, tai sotki pitkään, koska jotenkin sen huomas, että kun aluks, aluks yritti niinkun käsittää sen näillä, niinkun, vertauskuvilla, niin sit se jäi, sit sitä ei käsittäny ollenkaan, koska mä koko ajan tiesin, että tässä ei oo kyse mistään koneesta, mä en oikein päässy siihen sisään et mitä tässä haetaan tässä, niinku sen konkretisoimisessa koneeks.
H (98): Mmm.
B (98): Ja tota niin... Huah, mitähän muuta?
B (100): No toi on oikeestaan ehkä se, niinku, selkein esimerkki. Mullei nyt tuu muuta mieleen.
H (101): Joo. Okei. Tota. Mennään sit näihin käsitteisiin. Niin tota, kerrotko ensin, mitä on kapselointi?

B (102): Kapselointi. Jaa-a, siis. Mä olen käsittänyt sen niin, en mä tiedä onko jotain virallista määritelmää, mut siis mä oon käsittänyt sen niin et se on siis olioiden yhteydessä puhutaan kapseloinnista, joka tarkoittaa sitä että niinkun tietty määrä dataa tai informaatiota kirjoitetaan sillä tavalla et siihen ei pääse enää käsiksi sitten muualta, muualta kohtaa ohjelmaa, kuin sen niinkun olion sisältä, ikään kuin, tai sieltä.
H (110): Joo.
B (111): Tai sen luokan sisältä.
H (111): Joo. Mihin se informaatio on tallennettuna?
B (112): Mihin se on tallennettuna? Nyt mä en os..., en ymmärrä nyt kysymystä.
H (114): Onks se siinä... Miten se näkyy se informaatio, esimerkiks, jos kirjoitetaan sitä ohjelmakoodia.
B (117): No siis se on niinkun, kirjoitetaan luokan sisään, tämmöseks...
H (119): Mmm.
B (120): No.
H (120): No tota, totatota. Entä mikä sitte on aksessori?
B (123): No, ne on ne... Luokan metodit joilla sitä, siihen tähän mun äsken puhumaani kapseloituun informaatioon pääsee sitten käsiks sen luokan ulkopuoleltakin, joilla sit, niin, joo...
H (127): Eli aksessorit on metodeja?
B (128): Joo.
H (128): Joo. Ku meillon nää määritteet public ja private, ni onks aksessorit enemmän jompaa kumpaa?
B (129): Aa, joo... Jaa...
H (130): Vai voiks ne olla kumpaa tahansa?
B (131): *huokaa* Mä en itse asiassa tiedä, jos niinkun, mä oon käsittäny et aksessoreiks sanotaan nimenomaan niitä metodeja, joihin pääsee käsiks ulkopuolelta, eli jotka on niitä public metodeja, mutta luokkaan voi kirjoittaa mun käsittääkseni kyllä ihan private metodejakin, mut sitä, kutsutaanko niitä sitte aksessoreiks, ni sitä mä en tiedä.
H (136): Joo. Tota. Entä sitte ku oliolla on jotain kenttiä, ni tän kapselointiperiaatteen mukaan ni onks ne kentät sillon public vai private? Vai onks sillä väliä?
B (140): Nyt täs päästään taas tähän terminologiseen ongelmaan. Mä en tiedä, muista, mitä kenttä tarkoittaa.
H (141): Eli sen olion muuttuja.
B (142): Aivan, se oli, joo. Tässon hyvä esimerkki tästä terminologisesta sekaannuksesta, ku on tottunu siihen että muuttuja on muuttuja, ni sit yhtäkkiä jossain tilanteessa asia, joka on ihan selvästi mun silmilleni muuttuja, ni ruvetaanki puhumaan täysin eri nimellä, sitä käyt..., niinkun, sanotaan kentäksi.
B (146): Mut joo, siis, siis nämä, niinkun, olion sisäiset muuttujat, on private muuttujia, tai kentät, ne on niinkun siis tätä kapseloitua informaatiota, mistä mä puhuin.
H (150): Joo. Voiks sitä kapseloitua informaatiota sijaita myös muualla kuin olion kentissä?
B (151): Enpä tiiä.
H (152): Et onks se se pääasiallinen, ainakin miten sä aattelet että sitä...?
B (153): No siis pääasiallisesti mä oon ajatellu et se, tai siis tähän asti ainakin kaikissa esimerkeissä ja harjoituksissa, mitä on tehty, niin se on pääasiallisesti ollu ne muuttujat, jotka on ne private-versiot, et sitten on, on kyllä, kentässä voi olla niinkun mä sanoin, käsittääkseni myös näitä private metodeja, jotka on sit myös sitä kapseloitua informaatiota, koska niihinkään ei pääse käsiks sitten sen ulkopuolelta.
H (159): Joo. Joo. Entä tota, kerrotko vielä kerran, et mikä on olio, noin yleisesti?
B (163): Jaa.
B (167): Mä en itseasiassa oikein tiedä että mikä se olio on, en... edelleen...
H (169): Jos esimerkiks ton kapseloinnin kautta ajattelee?
B (169): No siis mä voin ajatellan sen niin että... Tai siis mä olen jotenkin kuvitellut, että se on niinkun tapa... No, puhuin siitä informaation kapseloinnista...
H (174): Mmm.
B (174): Niin tapa, tapa kapseloida informaatiota ja saada sitä sitte uudelleen käyttöön.
H (177): Miten ne metodit liittyy sit siihen asiaan?

B (178): Siis ne olion metodit?
H (178): Mmm.
B (179): No siis, no aksessorit on niitä, joita... Joiden avulla sen olion sisälle kapseloituun informaatioon pääsee käsiksi, joilla sitä voi muuttaa.
H (182): Mmm. Joo. Okei. Tota. Entä sit mikä, mitä tarkoittaa oliotyyppinen muuttuja?
B (187): Prrr... No se on siis muuttuja, jonka tyyppi on, tai siis se on muuttuja, joka voi saada arvokseen ainoastaan sen olion, jonka tyyppi se on, tai sen tyyppisen olion.
H (192): Mmm. Joo. Liittyyks tähän sit tommonen sana kun null?
B (194): Juu, kyllä varmasti, mutta en osaa selittää...
H (195): Suunnilleen et mitä se tarkoittaa?
B (196): Ei pienintäkään hajua.
H (196): Okei, tota, entä, mitä tarkoittaa alkeistyyppi?
B (198): Enpä tiedä sitäkään.
H (199): Mmm. Oisko esimerkiks, öö, jos aatellaan et meillon vaikka int ja sit meillon joku olio, niin kumpi niistä saattais olla alkeistyyppi?
B (202): No se int on alkeistyyppi, mut en mä osaa, siis, sitä, sen tarkemmin määrittää, tai siis, en pysty selittämään, mitä se tarkoittaa. Tiedän, että jotkut tyypit on alkeistyyppejä ja jotkut tyypit ei, mutta se mikä se ero on, niin enpä tiiä.
H (207): Mmm. Okei. Tota, entä mikä on olion tila?
B (208): Ei hajuakaan, tää tila on mulle aina ollu ihan käsittämätön juttu.
H (210): Mmm. Tota, ku puhutaan tilasta, esimerkiks näin algoritmin tilana, tai olion tilana, niin ajatteleksä että, onks siin kyse sillon semmosesta käytettävissä olevasta tilasta, vähän niinku englannin "space"...
B (214): Ei. Ei. Kun siis... Siis...
H (215): Vaan millasest on kyse?
B (215): Siis... Tila... No miten sen nyt sanois...
H (216): Onks se enemmän "tilanne"?
B (216): No, tilanne, joo. Siis, se, siinä, siinä sanamuodossa, tai siis merkityksessä.
H (218): Joo, joo, eli vähän niinku englannin "state"?
B (219): Joo, juuri, sitä hain, kyllä.
H (219): Joo, okei. Tota, totatota. Entä mitä tarkoittaa arvon palauttaminen?
B (222): Heh. Se tarkoittaa sitä, että palautetaan arvo. Emmä pysty selittää sitä. Tai siis en osaa.
H (223): Mmm. Mmm. Mihin asiaan se liittyy?
B (225): Metodeihin, käsittääkseni. Metodi voi olla arvon palauttava tai sitten ei.
H (226): Joo. Jos se ei oo arvon palauttava, ni miten se merkitään?
B (228): Miten se merkitään? No siis sinne metodin alkulitaniaan kirjotetaan void.
H (229): Joo. Tota. Totatota. Sit ku palautetaan arvo, niin millasella lauseella se siellä koodissa tehdään?
B (231): Return.
H (232): Joo. Onks se aina metodin lopussa se return-lause?
B (233): No siis periaatteessa siinä mielessä et se lopettaa sen metodin suorittamisen, ku siihen tullaan. Siinä mielessä se on lopussa, mut siis fyysisesti se ei välttämättä oo sen koodin lopussa.
H (236): Mmm. Joo. Totatota.
B (238): Tai siis niitä lauseita voi olla useampia siinä...
H (239): Joo.
B (239): Yhdessä metodissa.
H (240): Okei. Tota. Voisiksä sit mainita muutaman esimerkin lauseesta, joka luo uuden olion? Tai voiksä kirjottaa?
B (242): Lauseesta, joka luo uuden olion?
H (243): Mmm.
B (243): Jaa.
B (248): No menee vähän hankalaks.
H (249): Mmm. Voiksä kertoo mitä sä mietit?
B (250): Että mitä toi sun kysymykses tarkottaa. Sitä mä mietin. Yritän saada selville, et mitähän mun pitäs tähän kirjottaa.

H (253): Mmm. Jos sä muistat, tai jos sä aattelet jotain konkreettista harjoitustehtävää, meilloli vaikka joku, mikä se nyt oli, se viljavarastotehtävä, esimerkiks.
B (257): Mmm.
H (257): Ni, ni siinä jossain kohtaa luotiin uusia olioita.
B (259): *puhisee*
B (261): No siis siinä käsittääkseni se miten mä oon, niissä, niissä harjoituksissa on tehty, tai esimerkeissä, niin on, ensin määritelty niinku muuttuja, tai siis muuttujan nimi ja tyyppi, ja se tyyppi on se sen luotavan olion tyyppi, ja sitten sen olion arvoks kirjotetaan että new ja sitten se, se olion nimi, tai mikskä sitä nyt sanotaan.
H (269): Joo. Joo. Voiksä kirjottaa jonkun esimerkin?
B (270): Hoohhh!
B (271): No jos nyt on vaikka, sanotaan että viljavarasto-olio, niin... Pannaan nyt sen nimeksi vaikka varasto. Siinon Viljavarasto, joka on sen tyyppi, ja varasto ja on yhtä kuin, tai siis, on yhtä kuin merkki...
H (277): Mmm.
B (277): Ja sitten...
B (279): new Viljavarasto.
H (279): Mmm. Miten se loppuu sitten se, tulisko sinne loppuun vielä joku?
B (281): Aa, joo, siis sinne tulee... Eikä tuu puolipistettä kun, noin, sulut.
H (283): Onks se nyt kokonainen lause, vai puuttuuks siit vielä jotain?
B (285): No siis täält... Tosiaan se tarvitsee sen puolipisteen sinne perään se...
H (286): Joo. Okei. Eli tää luo uuden olion?
B (287): Joo.
H (288): Joo. Voiksä kertoo viel uudestaan, et mitä nää osat tarkoittaa?
B (289): Viljavarasto, se ensimmäinen viljavarasto, on se uuden muuttujan tyyppi. Varasto on sen muuttujan nimi, ja sitte se yhtäläisyysmerkki tarkoittaa et mitä se muuttuja saa arvokseen, ja sitten new Viljavarasto on se komento, millä se uusi mu... olio luodaan.
H (295): Mmm.
B (295): Se uusi viljavarasto-olio.
H (296): Mmm. Entä noi, nää sulut tässä, mitä ne tarkottaa?
B (297): Am, jaa-a, se onki hyvä kysymys. Se on... Jos sillä ois jotain parametrejä, ne kirjotettais siihen.
H (300): Mmm. Ää. Mille ne parametrit sitte menee? Mikä ne käsittelee?
B (303): Mille?
H (303): Mmm.
B (303): En... tiedä.
H (304): Onks tota konstruktori tuttu asia?
B (306): J... On. Ilmeisesti ne menee sitten sille, mut en oo kyllä koskaan ajatellut asiaa. En voi sanoa, että tietäisin.
H (309): Joo. Entä oisko sul mieles muita asioita, tai, tai muita esimerkkejä lauseesta, joka luo uuden olion? Jos mietitään et ihan aika alussa oli jo puhe näistä stringeistä, että string on aina olio, niin tota, voisko sen perusteella keksiä toisenlaisen esimerkin lauseesta, joka luo uuden olion?
B (317): Hmm.
B (322): No varmaan jos, jos mä nyt teen, tai siis esittelen string-tyyppisen muuttujan, niin se luo...
H (325): Mmm.
B (325): Luo sitten string-olion.
H (326): Luoks se sen sillon kun se muuttuja esitellään, eli jos sä sanot ihan vaan että "String jono;", siinähän se niinku esitellään se muuttuja...
B (330): Mun käsittääkseni, nyt menee kyllä arvauksen puolelle, mut mun käsittääkseni se luo sen olion silloin ku sille muuttujalle annetaan joku arvo.
H (332): Joo.
B (332): Tai se saa jonkun arvon.
H (333): Joo. Eli jos me sanotaan vaikka ensin että String jono1 on ja sit jotain, lainausmerkeis tekstiä...

B (336): Niin sillon se luo siinä vaiheessa, sen on-merkin jälkeen, ehkä, vähän vaikea sanoa, mut siinä vaiheessa sen olion.
H (340): Joo. Tota, entä sit jos me tehään semmonen sijoitus, että String jono2 saa arvokseen olio1, ni luodaaks sillon myös uus olio?
B (345): Joo.
H (345): Eli, eli sillon luodaan toinen olio, jonka sisältö on sama kuin sen ensimmäisen?
B (348): Hmmmm. En tiedä.
H (350): Vai oisko sen sisältö jotain muuta?
B (352): Nyt mä putosin jossain vaiheessa, heh.
H (353): Jos mä saan kirjottaa tän, tän mitä mä ajattelin.
B (353): Joo.
H (354): Eli meillon String jono1 saa arvokseen tekstiä lainausmerkeissä. Joo. Ja sit meillon String jono2 saa arvokseen jono1. Niin, ni luoks tää jälkimmäinen lause uuden olion?
B (364): Vaikee sanoa. Kyllä se mun käsittääkseni luo.
H (365): Joo. Eli sillon se, se lois uuden olion, jonka sisältö on sama ku toi jono1.
B (368): Joo.
H (368): Okei. Tota. Sit mullois toinen aihe. Tota. Ensinnäki, ooksä tehny, ku sä teet harjotustehtävii, ni sä teet ne koneella, ja käännät ja testaat?
B (373): Joo.
H (373): Joo. Ni tota, jos sul tulee sellanen tilanne, että ohjelma ei toimi niinku sä haluisit, tai niinku sä kuvittelisit et se toimis, niin, niin mitä sä sillon teet?
B (377): Siis tarkotat sitä, että se on, mä oon saanu sen käännettyy, mutta se ei toimi niinku mä haluisin?
H (378): Joo, eli se on käännetty.
B (379): Aa. No.
B (381): Katson sen kohdan missä se ei toimi, niinkun, niinkun mä haluaisin. Ja sit mä sieltä koodista etsin sen, sen vastaavan kohdan, eli siis missä...
B (386): Sen, sen kohdan joka saa aikaan sen toiminnan, mikä on eri, kun minkä mä halusin, ja rupeen katsomaan sitä, että missä mä olen ajatellut asian väärin.
B (390): Jos ei se löydy sieltä, ni sit mä siirryn muihin kohtiin.
H (391): Joo. Tota. Ooksä koskaan tehny tällasta, ää, niinku kynällä ja paperilla simuloinu jonku algoritmin toimintaa?
B (394): Olen, joo, montaki kertaa.
H (395): Ihan noiden harjoitustehtävien yhteydessä?
B (396): Joo.
H (397): Voiksä kertoo suunnilleen et miten sä sitä teet sitä simulointia?
B (398): Miten mä teen? No siis mä kirjotan sen kirjottamani algoritmin käsin, kokonaan paperille, ja sen jälkeen mä rupeen ääneen, ääneen puhumaan, ikään kuin käymään sitä kirjoittamaani algoritmia läpi, että nyt tässä tapahtuu näin ja näin, ja nyt tässä tapahtuu näin ja näin, ja sitten jos sieltä tulee jotain arvoja tai esimerkiks numeroarvot muuttuu, ni mä kirjotan ne ylös sitte peräkkäin.
H (408): Joo. Okei. Joo. Onks sul muita keinoja, mitä sä oisit käyttäny virheiden etsimiseen?
B (411): Mmmh.
B (413): Totatota.
B (414): No siis, mä oon tehny sitä, et mä niinkun muutan siinä algoritminpätkässä, tai algoritmissa, joka toimii väärin, niin sanotusti, niin tota, muutan siinä jotain kohtaa jotenkin radikaalisti, ja kokeilen et miten se toimii sit. Että niinkun katsoakseni, että mitkä, kun et onko se juuri se kohta algoritmia, jonka mä kuvittelen vaikuttavan siihen, mitä se ohjelma tekee juuri siinä kohtaa, ni onko se se.
H (428): Mmm.
B (429): Esimerkiksi näin.
H (432): Okei.
B (432): Ja sitten mä yritän myös, yleensä mä alotan siitä et mä tarkistan kaikki sulut ja puolipisteet ja sulkeet sieltä koska, vaikka se meniskin kääntäjästä läpi ni monta kertaa se virhe on ollu siellä et mullon se puolipiste väärässä paikassa tai se yks kaarisulku liian vähän tai liikaa, jolloin se tekee ihan muita juttuja ku mitä sen pitäis.

H (441): Joo. Eli tota. Nää, nää kaarisulkeet, millä siis erotetaan tällanen lohko, eli lauseiden joukko, ni tota, niillä jäsennetään sen ohjelman rakennetta...
B (447): Joo, kyllä.
H (448): Ni tota, niiden käyttö ei oo vielä ihan automatisoitunu?
B (450): No, on se siis, en tiedä, mikään tässä ohjelmoinnissa ei oo vielä automatisoitunu mulla niinku kahden ja puolen viikon jälkeen, se on ihan turha ees ajatella.
B (454): Kaikki asiat mitä mä teen ni täytyy kyllä, mikään ei tuu selkärangasta, kaikki täytyy miettiä.
H (456): Mmm.
B (456): Ja mut tota, kyllä ne kaarisulut yleensä tulee laitettua, mutta usein siinä käy niin että varsinkin jos niitä on paljon, jos on monta eri lohkoa peräkkäin, jotka pitää, niin sit sieltä unohtuu joku, tai sitten katson väärin, et mä olin jo kirjoittanu sen sinne, vaikka mä en sitten loppujen lopuks olekaan.
B (463): Varsinki jos sisennykset on menny huonosti tai unohtunu, niin, sitten omat silmäni sotkevat.
H (467): Mmm. Tota. Voiksä kertoo vielä näist sisennysten käytöstä, et miten sä yrität käyt..., tai, tai käytät, tai pyrit käyttämään niitä?
B (471): No siis. Mmmh. Ikään kuin aina, no, jos nyt käytetään vaikka tota lohkoa, niin lohkot erotetaan sisennyksillä. Eli jos on lohko jonka sisällä on toinen lohko, niin ne sisennetään aina enemmän sitä mukaa, ja sit, sit niinkun, niin, esimerkiks näin.
H (483): Joo. Joo. Okei. Tota. Mennään sitte, mä näytän sulle vähän koodia. Ooksä näit taulukoita ehtiny? Niin, viimekerran tehtävät sä sanoit tehnees, niissähän nää tuli...
B (489): Joo.
H (489): Joo. Ni tota, kerroksä nyt et mistä tässä on kyse?
B (492): Mm-hmm-hmm.
H (493): Jos sä voit kertoo ääneen sen mitä sä mietit.
B (494): Mietin että enpäs tästä nyt pysty sanomaan, mistä tässä on kyse. Jotain tässä taulukoita käydään, käydään läpi...
H (497): Mmm. Jos sä lähet ihan jossain, jossain järjestyksessä selvittämään.
B (500): No, tässä onki sitte selvitystä.
B (503): (nauhan puoli loppui)
B (505): Niin, siis tässä on...
H (506): Mmm.
B (506): En mä nyt pääse tähän taas käsiksi ollenkaan. Tässon jotain tekemistä taulukkojen kanssa, tässä on.
H (508): Mmm.
B (508): Siinä ensin tehdään muuttujia, int-tyyppisiä muuttujia, mihinAsti, parhaillaan ja apu. Ja mihinAsti saa arvokseen luvut.length - 2, josta en ymmärrä mitä se tarkoittaa, parhaillaan saa arvokseen 0, ja apulle ei vielä anneta arvoa.
H (512): Mmm. Ää. Mikä toi luvut olis, löytyyks se täältä jostain, siihen vastaus?
B (515): Se on tuolla... Parametrina.
H (516): Joo. Eli mikä se on?
B (516): Mikä se on?
H (516): Nii, vaikka tyypiltään?
B (517): Se on... Taulukko, int-taulukko, ilmeisesti.
H (519): Joo, joo. No jos me katotaan tota, tätä metodin määrittelyä, tää rivin public static int hakasulut ja niin edelleen, niin mitä kaikkee toi rivi sulle kertoo?
B (522): Hmm, hmm. No, se on public-metodi, ei oo private. Sit siinä on int ja hakasulut, joka tarkoittaa, että se palauttaa arvon, jonka tyyppi on int-tyyppinen taulukko, sen arvon. Sit on teeJotain, joka on sen nimi, ja sen jälkeen on suluissa parametri.
H (528): Mmm. Eli se saa yhden parametrin?
B (528): Joo.
H (528): Jonka tyyppi on int-taulukko.
B (529): Joo.

H (529): Joo. Eli nyt jos me katotaan tota, tota int mihinAsti -riviä, ni me tiedetään ainaki että luvut on int-tyyppinen taulukko. Muistaksä mitä toi .length vois tarkottaa?
B (532): No siis se tarkottaa sen taulukon pituutta, niin sanotusti, mut se mitä se niinkun konkreettisesti tarkottaa, ni mullei oo mitään haisua.
H (535): Mmm.
B (535): Mä ajattelen sen niin että se tarkoittaa sen taulukon pituutta, mutta siihen se jääkin.
H (536): Eli sä et osaa nyt sijoittaa sille mitään tiettyä lukuarvoa?
B (537): Mmm, en, siis, mitään tiettyä lukuarvoa.
H (538): Joo. Olisko se mahdollista tän koodin perusteella, yleensäkään?
B (539): Siis tämän, mitä tässä nyt nää kolme ensimmäistä riviä, mitä ollaan käyty läpi?
H (540): Nii, nii, esimerkiks?
B (541): *huokaa* No mun käsittääkseni ei, koska tässä ei missään vaiheessa käy ilmi, että minkä pituinen se taulukko on, se pitäs ensin määrittää.
H (543): Joo. Mutta että, mut et tää muuttuja mihinAsti saa arvokseen luvun, joka on, on suuruudeltaan tän taulukon alkioiden määrä miinus kaksi.
B (547): Hmm, mmm. Joo.
H (547): Joo. Okei. Tota. Eli toi luvut on nyt tälle metodille, on tällanen muodollinen parametri, eiks niin?
B (549): Joo.
H (549): Ja sitte kun tätä metodia kutsutaan, niin, ni siihen sijoitetaan joku todellinen parametri...
B (551): Kyllä.
H (551): Joka on joku taulukko. Okei. Eli me ei nyt vielä tiedetä, et mikä se, se todellinen taulukko on, mitä sit käsitellään, mut me voidaan tutkii silti, et mitä tää metodi tekee.
B (553): Mm-hmm.
H (554): Joo. Ja jos, jos jatketaan sit siitä eteenpäin.
B (555): Sitten tulee while-lause, jossa ehtona on että tämä int-muuttuja mihinAsti, jonka arvo oli siis tän taulukon pituus miinus kaks, ni sen pitää olla suurempi tai yhtäsuuri ku nolla.
H (558): Mmm.
B (559): Sitten tämä parhaillaan-muuttuja saa uudestaan arvokseen nolla, syystä jota en ymmärrä.
B (561): Aa, sit tulee toinen lause, eikun siis toinen while-lause, jossa ehtona on että tämän parhaillaan pitää olla pienempi tai yhtäsuuri kuin mihinAsti-muuttujan...
H (564): Mmm.
B (564): Eli siis tämän taulukon pituuden miinus kaksi.
H (567): Mmm.
B (567): *huokaa* Ja sen alla tulee if-lause, josta mä en sit ymmärräkään enää mitään.
H (569): Mmm. Eli siellon joku asia, siellon toi taulukon nimi, ja sit siellon hakasuluissa se joku asia. Siellon...
B (572): Kyllä, siellon hakasuluissa parhaillaan.
H (573): Joo. Joka on muuttuja, eiks niin?
B (573): On.
H (573): Joo. Ää, eli toi ei oo sulle tuttu tää, tää ilmaisu et mitä tää tarkoittaa et ku taulukon perässä on hakasulut ja siellä jotain?
B (576): En mä voi sanoa etteikö se olis tuttu, kyllä mä olen sen tyyppisiä tehtäviäkin tehny, joissa tätä käytetään, mut...
H (577): Mmm. Osaisiksä sanoa, mihin se viittaa?
B (578): *huokaa* Siis...
B (580): No se viittaa, no siihen alkioon taulukossa, jonka järjestysluku on toi parhaillaan, tai siis, niin...
H (583): Joo, joo. Ihan täsmälleen. Tota, ää, muistaksä vielä et mikä on taulukon ensimmäisen luvun, tää järjest...?
B (586): Nolla.
H (586): Nolla, ja siit eteenpäin. Ja viim..., jos ajatellaan, et toi length kertoo jonkun luvun, ni mikä silloin on aina taulukon viimeisen alkion järjestysluku?
B (589): Se on se length-1, yhtä pienempi.
H (590): Joo. Joo. Okei. Eli osaisiksä nyt tän perusteella kattoo tätä if-lausetta?

B (593): *huokaa* No en. En osaa sen paremmin.
H (596): No entä noin yleisesti jos sä katot tätä metodia niin tota, niin niin, niin niin, osaaksä jotain yleisiä asioita sanoo tästä, et mitä tästä näkis päälle?
B (600): Eeeh, mitä tästä näkis päälle, yleisiä asioita?
H (601): Mmm.
B (601): En nyt oikeen. En tiedä mitä sä nyt tarkotat, mut en mä tästä nyt kovin paljoa nää. Näen sen että tää on arvon palauttava metodi ja se palauttaa jonkun arvon.
H (603): Mmm. Ja siinon, missä paikoissa siellon palautusarv..., tai palautuslauseita?
B (605): Siellon ihan viimeisenä on yks return.
H (606): Mmm. Eli siel, siel ei oo muita?
B (606): Ei.
H (606): Joo. Eli siitä, sillon se metodi suoritetaan aina tietyssä järjestyksessä, tänne loppuun saakka. Eiks niin?
B (609): Joo, kyllä.
H (609): Et se suoritus ei voi loppuu muussa kohti?
B (610): Ei.
H (610): Joo. Entä millasii toistolauseita siellä olis?
B (610): Siellä on kaks kappaletta while-lausetta.
H (611): Mmm.
B (611): Ja yksi kappale... Ei, ei siellä ole muita.
H (612): Joo.
B (613): Ja while-lauseet on siis niitä, että niille annetaan se ehto, ja sitten se, joka kerran se alussa tarkistetaan, et pitääkö se paikkansa, ja sen jälkeen kaikki sen lohkon sisällä olevat asiat suoritetaan niin kauan, kunnes se ei pidä enää paikkaansa.
H (616): Joo. Tota, totatota. Entä jos katotaan tota sisempää while-lausetta, ni voiksä vaik merkitä et mitkä kaikki lauseet kuuluu sen while-lauseen toistettaviin lauseisiin?
B (620): Sen while-lauseen toistettaviin lauseisiin kuuluu... Tässä.
H (623): Noin. Joo. Eli se if-lause ja sit siinä...
B (623): If-lause ja sit siinä on vielä yks.
H (624): Joo. Entä sit sen ulomman while-lauseen toistettaviin lauseisiin?
B (625): *huokaa* Siihen kuuluu tämä, tän äskeisen while-lauseen lisäksi... Hetkinen, nyt meni. Odotas nyt.
H (629): Mmm. Eli kuuluuks se mihinAsti...?
B (629): Eli, mmm, mmm, mul, nyt tää menee siihen et mun täytyy nyt vähän laskeskella, mulla hyppii nää aaltosulut silmissä aina, niinkuin mä sanoin tossa aikaisemmin.
H (632): Auttasko jos sä vaikka numeroit noit aaltosulkupareja sinne paperille?
B (633): Joo, varmasti auttais. Jos mä nyt jätän noi alkupään aaltosulkuparit numeroimatta.
H (635): Joo. Eli tosson noi parit. Joo. Ja se mihinAsti = mihinAsti - 1, se tuli nyt sinne sisään.
B (637): Niin se tulee tän sisään, joo, kyllä.
H (637): Joo, okei. Joo. Joo. No jos mä näytän sitte toisen koodinpätkän, joka tavallaan liittyy tohon äskeiseen.
B (641): *huokaa* Hmm.
H (642): Kerroksä taas et mistä on kyse?
B (643): No, siinä on tämmönen asia, jota yleensä harjoituksissa ja muuten kutsutaan pääohjelmaksi, en tiedä miksi. Mutta siinä on rimpsu public static void main String ja niin edelleen, args. *huokaa*
H (646): Mmm.
B (647): Sitten siinä luodaan uusi taulukko. Ja niin kuin tässä kommentissa sanotaan, jonka sisällöksi sijoitetaan luetellut luvut.
H (649): Mmm.
B (649): Siellä on kuusi, seitsemän lukua.
H (650): Mmm.
B (650): *viheltää* Sit meneeki hankalaks.
H (652): Mmm. Ni sit, sit esitellään toinen muuttuja...
B (653): Esitellään toinen, toinen taulukkomuuttuja.

H (653): Joo. Ja sen arvoks sijoitetaan jotain.
B (655): Joo.
H (655): Joo.
B (655): Niin tehdään.
H (656): Nii. Onks tää TekeeJotain, tolleen isolla alkukirjaimella tossa niin, niin mihin se viittais?
B (659): No. Empä tiiä. Se on sama asia, kuin mikä lukee tässä aikasemman koodinpätkän alussa, public class TekeeJotain.
H (661): Eli onks se sen luokan nimi?
B (662): *huokaa* Jaa-a, ilmeisesti.
H (663): Joo. No, okei. Eli meillon tota, täähän on static metodi, eli se on niinku sen luokan metodi, eli sitä voi kutsuu sen luokan nimen kautta.
B (666): Jaa-a, selvä.
H (666): Tää, tää ei ehkä oo tullu luennolla...
B (667): Ei, tää on ihan uutta.
H (667): Mutta voin sen verran tässä, tässä auttaa. Niin tota, totatota, joo. Eli siinon kutsutaan tätä teeJotain-metodia, mitä sä äsken katoit.
B (670): Ja sit se saa parametrikseen ton taulukon, joka on siis se uus, äsken tehty uus taulukko.
H (672): Joo. Ja tota... Okei. Tota. Nyt, nyt sullon sit ainaki annettu tää taulukko. Pystyisit varmaan ainaki piirtämään sen, että, et miltä se näyttää ja mitä siellon sisällä.
B (676): Piirtämään?
H (677): Niin. Tai kirjottamaan.
B (678): *huokaa* No. Piirretään se nyt vaikka tämmösenä palkkina, ku nämä on luennolta tuttuja.
H (680): Mmm.
B (680): Siinä on, hetkinen, montako niitä oli, seittemän lukua.
B (683): Eli se on taulukko, jossa on, tai siis palkki, jossa on seittemän, joka on jaettu seittemään osaan, ja joka osassa on, joka osaan mä kirjotin sen, sen arvon mikä siinä...
H (687): Mmm. Entä sitte nää, nää indeksit, eli järjestysnumerot, voisiksä kirjottaa ne vaik...
B (688): Nn, joo, mä kirjotan ne vaikka tonne palkin yläpuolelle. Tosta, nollasta kutoseen.
H (690): Joo. Okei. Eli nyt, nyt tää teeJotain-metodi saa tällasen taulukon todelliseks parametrikseen, eli...
B (692): Kyllä.
H (693): Nyt meillä periaatteessa olis kaikki täsmälliset tiedot käytettävissä niin et me voitais simuloida sen toimintaa.
B (695): Jaa?
H (696): Ni voisiksä nyt vähän aikaa simuloida?
B (696): No minäpä simuloin sitten.
B (699): Tässä ensin tämä mihinAsti-muuttuja saa arvokseen sen taulukon pituuden miinus kaks, eli tässä tapauksessa sen arvoks tulee viis.
H (702): Mmm.
B (703): Ja sitten parhaillaan on nolla. Ja apu ei ole vielä mitään.
H (705): Mmm. Joo.
B (705): Mä kirjotan tähän lyhennettynä nää muuttujien nimet.
B (706): Sitte mennään sinne while-lauseeseen ja tarkistetaan onko mihinAsti suurempi kuin nolla, ja se on tällä hetkellä viisi, eli se suoritetaan se, sen lauseen sisältö, tai siellä olevat jutut.
B (710): Parhaillaan on edelleen nolla, tai siis saa arvokseen uudestaan nolla, mitä mä en edelleenkään oikein ymmärrä.
H (712): Mmm.
B (712): Mutta sitten siirrytään seuraavaan while-lauseeseen, jonka ehto on että parhaillaan pitää olla pienempi tai yhtäsuuri kuin mihinAsti, ja sehän pitää paikkansa, koska se on nolla ja mihinAsti on viis, eli suoritetaan se, sisältö.
H (716): Mmm.
B (716): Siinä on if-lause, että, jossa jossa. Siis sen nollakohdassa, taulukon nollakohdassa olevan arvon pitää olla suurempi kuin taulukon ykköskohdassa oleva arvo.
H (722): Mmm. Eli mitkä ne arvot on?
B (723): Se ensimmäinen on seitsemän ja toinen on kakstoista. Eli se ei siis pidä paikkaansa.

H (724): Joo.
B (725): Jolloin tätä if-lauseen, tai siis se suoritus loppuu siihen, että ei suoriteta sitä.
B (727): Sit käydään mitä siellä sisällä on.
B (728): Ja sitten siirrytään suoraan sinne while-lauseen loppupuolelle, jossa on parhaillaan saa arvokseen parhaillaan plus yksi, eli se on nyt sitten yksi, sen arvo.
H (730): Mmm.
B (730): *huokaa* Ja sitten mennään taas tarkistamaan sen while-lauseen ehto.
H (732): Mmm.
B (732): Ja se on edelleen pienempi kuin mihinAsti se parhaillaan, koska se on nyt yks ja mihinAsti on edelleen viisi.
B (734): Haah. Sitte katsotaan se if-lause.
B (736): Eli nyt pitää olla ykköspaikalla olevan luvun suurempi kuin kakkospaikalla olevan luvun. Se pitää paikkansa, koska ykköspaikalla on kakstoista ja kakkospaikalla on kakkonen.
B (739): Jolloin suoritetaan se mitä siinä if-lauseen alla on siinä, muuttuja apu saa arvokseen, hetkinen, sen parhaillaan-paikassa olevan arvon, eli ykköskohdassa olevan arvon, eli kakstoista.
H (745): Mmm.
B (745): Sitten...
B (748): Sit se kakstoista saa uudeksi arvokseen tästä arvon, joka on ykköspaikalla, eikun siis plus yksi, eli kakkospaikalla, eli kakkosen.
H (753): Mmm. Pitäskö sun sekin kirjottaa. Varmaan tonne, jos sä jatkat siihen alle, niin...
B (757): Niin. Siis hetkinen, mitä mä nyt sanoin, nyt mä sotkeennuin.
H (758): Et se ykköspaikalla oleva...
B (758): Ykköspaikalla oleva saa arvokseen siis sen joka on kakkospaikalla. Eli siitä tulee silloin kakkonen sen kahdentoista sijaan. Itseasiassa mä laitan sen tonne.
H (762): Joo.
B (762): Ja sitten, sitten se kakkospaikka saa arvokseen sen avun, eli kakstoista. Eli ykkönen ja kakkonen vaihto paikkaa.
H (765): Joo. Okei. Ja sitte tapahtuu?
B (766): Hmm, sitten mennään taas siihen, siis sit se if-lauseen suoritus, tai siis se, osan suoritus loppuu, ja mennään siihen while-lauseen loppupuolelle, jossa parhaillaan, joka on tällä hetkellä yks, saa arvokseen taas plus yksi, eli siitä tulee kaksi.
B (775): Ja näin sitten edetään. Mennään taas uudestaan sinne while-lauseeseen, ehtoon, siinä on, se on, parhaillaan-arvo on kaksi eli se on pienempi kuin miinus viisi, eli sitten katsotaan tämä if-lause, kaksi seuraavaa.
H (780): Mmm.
B (780): Joista kakkosen arvo on kakstoista ja kolmosen arvo on yhdeksän, joten niiden paikkoja pitää vaihtaa.
H (784): Joo. Voiksä kirjata senki sinne?
B (785): Voin. Eli siis apu saa arvokseen kaks...
H (787): Hetkinen, meilloli kakkosen ja kolmosen...
B (787): Eikun, kakstoista, anteeks. Mä sotken, sotken itseäni.
B (789): Niin siis apu saa arvokseen kaks...
H (790): Eikun kakstoista.
B (790): Eikun kakstoista, niin. Ja sitten tota niin, tämä kakstoista saa arvokseen kolmosen arvon, eli siit tulee yhdeksän.
B (794): Ja sitten se kolmonen saa arvokseen sen avun arvon eli siit tulee kakstoista, eli ne vaihto taas paikkaa.
B (796): Ja sitten se parhaillaan-arvo kasvaa yhdellä, eli siit tulee kolmonen.
H (798): Mmm.
B (798): Sitten verrataan taas, mennään sinne while-lauseen alkuun, niin se edelleen, ehto pitää paikkansa. Sitten verrataan kolmos- ja nelospaikkoja. Ja taas pitää vaihtaa niiden paikat, koska kolmosen arvo on isompi kuin nelosen arvo, eli...
H (803): Niinku sä iteki sanoit et niiden paikka vaihtuu, ni tarviiksä nyt, jos sä ajattelet sitä paikan vaihtoo, ni tarviiksä tätä apu-muuttujaa välttämättä?
B (807): Siis ajattelussani, vai?

H (807): Niin, tai, tai tarvii... Jos sä niinku haluut vaan simuloida tätä toimintaa...
B (808): Siis jos mä haluan simuloida sitä niin että mä pysyn kokoajan ite kärryillä siitä mitä mä oon tekemässä, niin kyllä mä tarviin.
H (810): Joo. Okei.
B (810): Et mä... Tää on niin, on, kokoajan täytyy, sen takia mä puhun kaikki nää asiat myös ääneen, koska mun täytyy joka kerta ku mä nään näitä lauseita, ni miettiä et mitäs tämä nyt tarkottikaan.
H (813): Joo. Joo.
B (813): Niin tota ni, samalla lailla mun täytyy kirjottaa tää apukin ylös, koska muuten mä tipahdan itse omasta simuloinnistani.
H (815): Joo. Sä kuitenki huomasit heti ekan kerran ku sä teit sen operaation et siinon kyse niiden paikkojen vaihtamisesta.
B (818): Joo. En tiedä olisinko huomannut jossei tätä olis käyty läpi luennoilla aika monen esimerkin kautta.
H (820): Joo. Okei.
B (821): Mut...
H (822): Joo. Eli nyt sä olit toteuttamassa sitä paikanvaihtoa, ku se parhaillaan oli kolme, eli...
B (824): Aivan. Niin. Eli siis. Eli apu saa sen kolmosen arvon, joka on kakstoista. Ja sitten ne vaihtaa paikkaa ne kolmonen ja nelonen, eli kolmosen arvoks tulee yhdeksän, ja nelosen... Eikun kolmosen arvoks tulee kolme ja nelosen arvoks tulee yhdeksän.

H (832): Mmm.
B (832): Ja sitten... Niin, aivan, näin. Ja sitten parhaillaan kasvaa taas yhdellä.
H (835): Mmm.
B (835): Siitä tulee neljä.
H (836): Mmm.
B (836): Ja sitten tehdään vielä kerran, sama juttu, verrataan nyt nelosta ja vitosta.
B (838): Ja koska se if-lauseen ehto ei toteudu, niin niitten paikkaa ei vaihdeta, nelonen on yhdeksän ja vitonen on kahdeksantoista, eli niitten paikat pysyy ennallaan.
H (841): Mmm. Mitä sit tapahtuu?
B (842): Ja sitten parhaillaan kasvaa yhdellä, siit tulee viisi. Ja sitten mennään taas sen while-lauseen alkuun, ja sitten verrataan vitosta ja kutosta, ja vitonen on isompi kuin kutonen, tai siis vitosen kahdeksantoista on isompi kuin kutosen nolla, joten niiden paikkaa vaihdetaan.
B (851): Ja sen avun arvo on nyt kahdeksantoista, ja sitten vitosen arvoks tulee nolla, kutosen arvoks tulee kahdeksantoista.
H (852): Mmm.
B (852): Ja sitten kasvatetaan taas parhaillaan-arvoa yhdellä, siitä tulee kuusi, ja sitten tämä sisemmän while-lauseen ehto ei enää täyty, niin että se loppuu, sen pyöritys, ja sitten mennään takaisin tän ulomman while-lauseen ehtoon.
H (858): Mmm.
B (858): Eikun ensin, ensin tehdään tää uloin while-lause ensin loppuun, eli mihinAsti saa arvokseen mihinAsti miinus yksi, eli neljä.
H (862): Joo. Osaisiksä nyt sanoo, että, et seuraavan kerran ku suoritetaan toi sisempi silmukka, näin käsitteellisesti, ni mitä sillon tehään?
B (866): Sillon käydään, kun tän ensimmäisen silmukan, tai ensimmäisen kerran ku tää while-silmukka käytiin läpi, siis tää sisempi, niin saatiin siirrettyä isoin näistä arvoista, mitä tässä taulukossa on, viimeseks.
H (872): Mmm.
B (872): Ja nyt käydään sitten kaikki ne, sitä viimestä lukuun ottamatta kaikki, muut viis arvoa.
H (875): Mmm.
B (875): Samalla tavalla läpi ku mitä mä äsken selostin, jonka tuloksena sitten taas tästä jonosta isoin saadaan sen jonon viimeseks.
H (879): Joo.
B (879): Ja niin sitte edetään loppuun saakka. Ja lopputulos on sit se, että ne on suuruusjärjestykseen järjestetty ne taulukon arvot.

H (883): Joo. Okei. Eli nyt sä sait selville et mitä tää metodi tekee.
B (884): Joo.
H (885): Joo. Eli sille annetaan... Voiksä kertoo ite?
B (886): Siis sille annetaan parametriks joku taulukko ja sitten se järjestää sen taulukon arvot, suuruusjärjestykseen pienimmästä suurimpaan.
H (892): Joo. Ja se palauttaa...?
B (893): Ja se palauttaa sen taulukon, järjestetyn taulukon arvonaan.
H (896): Mmm. Onks se se sama taulukko, vai onks tää luonu uuden taulukon?
B (898): Se on se sama taulukko.
H (898): Joo. Okei.
B (899): Mutta uudessa muodossa.
H (900): Joo. Okei. Tota, totatota. Tässolis nyt tällä kertaa kysymykset. Toi aikakin loppuu ihan kohta. Sopisko jos vielä sovitaan uus aika.
B (905): Joo, nyt voisi sopia sen.
H (906): Tai itseasias meillon kyl sovittunaki.
B (907): Niin mä muistelisin, se oli ens viikon keskiviikkohan se oli. Eikö niin?
H (910): Hetkinen...
B (910): Oliks meillä? Eiku mitä? Nyt mä oon... Hetkinen... Missä viikossa me ollaan?
H (912): Ensviikko alkaa yheksännestä päivästä.
B (913): Yheksännestä päivästä. Niin, milles päivälle meilloli sovittu se?
H (916): Kahestoista päivä.
B (916): Ah, okei, mä oon kirjottanu sen tänne yhennelletoista päivälle. No, hyvä että tuli puheeks.
H (917): Joo. Eli kahestoista päivä ja kakstoista kolkyt. Tuunks mä hakeen tuolta ala-aulasta?
B (920): No enköhän mä osaa tulla tänne, jos sä täällä oot.
H (920): Joo. Joo. Okei. Ni tota. Joo. Haluuksä et mä viel muistutan tosta jollain tavalla, tai...?
B (923): Kyl mä nyt varmaan, ku mä kerran tän unohdin, ni kyl mä varmaan nyt osaan muistaa ne.
H (926): Joo. Okei. Kiitoksia taas tosi paljon. Yks asia ainakin mun piti sanoo, ettei jää väärä käsitys, ku mä tivasin tätä, näit jonojuttuja.
B (930): Joo.
H (930): Ni tota, niin, niin tässä ei luoda uutta olioo.
B (932): Jaaha.
H (932): Se pitää korjata, eli, eli tässä niin, tää jono2 ni se saa viitteen siihen samaan olioon, eli meillon kaks muuttujaa, jotka molemmat viittaa samaan olioon, näitä olioita on vaan yks.
B (937): Ahaa. Niinkö se meni? Niinkö se meni? Selvä.
H (938): Joo. Joo. Halusin, ettei jää väärä käsitys.
B (939): Mm-hmm.
H (940): Joo. Okei. Kiitoksia.